

Journaling: A Look At The
OS X Personal Diary

Extending
Spotlight's Reach

Building A Cocoa
Screen Saver

MacTech Magazine

June • 2005

MACTECH[®]

The Journal of Macintosh Technology

**Tons Of Great Content That Is
Sure To Thrill And Amaze You!**

**PHP And MySQL,
Together At Last**

User Interface Scripting

Life With SQLite

**Developing Applications With
The QuickTime For Cocoa Kit,
Part II**

**launchd: Judge,
Jury, Executioner**

**The Heart & Soul
of Open Source:
Drunkenbatman and
The Source Hound**

\$8.95 US
\$12.95 Canada
ISSN 1067-8399
Printed in U.S.A.



Need to find something fast?

INDEX INDEX INDEX

With c-tree Speed.

**TRY IT
NOW!**



FairCom's c-tree Plus®

embedded database engine offers Superior Indexing Technology – the key to performance, data integrity, and concurrency. c-tree Plus offers direct record-oriented C and C++ APIs with an industry-standard SQL interface that allows use of any combination of APIs within the same application. Furthermore, we offer source code access for intimate programming control, unmatched portability, and developer-to-developer technical support.

Heterogeneous Environments

c-tree Plus and c-treeSQL™ Servers are the perfect solution for your mixed platform environments. Mac servers to Windows clients? No problem. Linux Servers to Mac clients? No problem. c-tree has a long history of cross-platform development solutions. Byte incompatibility between platforms is handled seamlessly with our Unifomat data handling technology.

Low TCO

c-tree is priced affordably, requires minimal hardware resources, and needs no IT staff for maintenance. If Total Cost of Ownership (TCO) is important to you, c-tree is the perfect database.

Easy Deployment

c-tree Servers are designed for ease of use and deployment as well. Out of the box, our servers can be installed and running in minutes.

Start indexing your data today!



FairCom®

USA • Europe • Japan • Brazil

www.faircom.com

Go to www.faircom.com/go/mteval for a **FREE** evaluation of c-tree Plus!

Worried about the switch to Intel?



Here is a sampling of our sessions:

Mac OS X

Inside Dashboard Widgets • Out of the Bottle: The Genie Effect • Techniques for Universally Working with Network or Local File Names and Paths Under Carbon

Cross-Platform

Supporting Intel Macs: The Easy Stuff and The Nitty Gritty • Cross-Platform Approaches from a Macintosh Perspective

Coding Practices

The Design of Everyday Code • Program Analysis and Verification on Mac OS X

Business

So you want to work at Google? • Successful Project Management • Hacking the Press

Come to the 20th Anniversary of ADHOC to learn from people with experience moving applications from PowerPC to the new Universal Binary Format. We'll have sessions covering the basics and the details of migrating your application, cross platform programming issues, and, of course, a wide range of other Mac programming and business topics. We will naturally be covering exciting Tiger Technologies so you can show off to all your friends.

You'll also have a chance to talk with experienced programmers who have already migrated their applications to run on Apple's Intel hardware.

Then there's our featured speakers, **Jordan Hubbard** and **Cory Doctorow**. Who knows what the God of BSD will tell us in the wee hours of the morning? (Well, technically Jordan is the Engineering Manager for BSD Technologies, but we know the truth.)

And, of course, there's the ADHOC Showcase. Hosted by beloved industry professional **Andy Ihnatko**, you'll get to show off what 48 hours of caffeine-induced programming frenzy will inspire you to create. Something wonderful? Something dangerous? Worst of all, possibly something useful? Don't be afraid to find out what strange things lurk in the back of your mind.

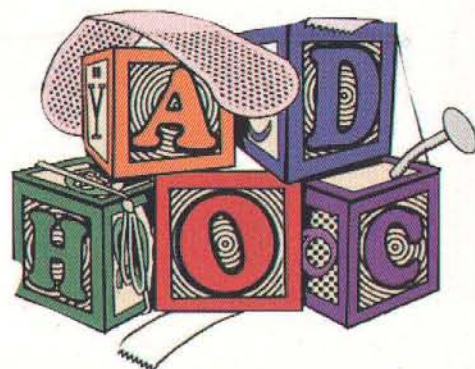
**Sign up for pre-conference
Applescript/Automator
Training with Matt Neuburg**

**Free copies of
REALbasic Pro
for all attendees!**

ADHOC 2005, July 27-31

For more information, and to register, go to

<http://www.adhocconference.com/>



Sponsored by Google • NVIDIA • REAL Software • Digital Forest • MacTech • O'Reilly • Speakeasy

ITANIUM, AND PENTIUM, AND MACTEL, OH MY!

Lucifer has just set up a snow cone stand down on the corner. The thing that would never happen, happened. Apple announced at the June Worldwide Developers Conference that, beginning in 2006, Macs will ship with Intel Inside. Has the chime begun to haunt your dreams yet?



All over the Mac web, and on a great many mailing lists, variations on "the sky is falling!" theme are being played out. But, why? Let's look at a few of the contentions of the complainers.

First, Intel is evil. They are the bad guys, in cahoots with Microsoft, they are SATAN! Meh. That's MacMac thinking, and has no place in reasoned discussion. Intel is a company that makes chips. IBM is a company that makes chips. Apple no longer sees IBM heading in the direction they need. Evidence the lack of a G5 PowerBook now and in the foreseeable future. The roadmap as given to Apple had both desktops running at over 3 GHz, and G5 PowerBooks in the stores today. It didn't happen, nor was it going to. The 970 roadmap took a left when Apple needed to go right. Intel is willing to meet Apple's needs, and it would seem, has shown Apple they can get there, while IBM cannot.

Second, the problem of rewriting code to run on Marklar. This is, for the most part, a nonstarter. Yes, there are going to be some

MACTECH

A publication of
XPLAIN CORPORATION

The Editorial Staff

Publisher

Neil Ticktin

Associate Publisher

David Sobsey

Editor-in-Chief

Dave Mark

Graphics & Production

Dennis Bower

Regular Columnists

Getting Started

by Dave Mark

QuickTime Toolkit

by Tim Monroe

Reviews/KoolTools

by Michael R. Harvey

Patch Panel

by John C. Welch

AppleScript Essentials

by Ben Waldie

The Source Hound

by Dean Shavit

Mac In The Shell

by Ed Marczak

Board of Advisors

Jordan Dea-Mattson, Jim Straus
and Jon Wiederspan

Contributing Editors

Michael Brian Bentley
Vicki Brown
Marshall Clow
John C. Daub
Tom Djajadiningrat
Andrew S. Downs
Gordon Garb
Chris Kilbourn
Rich Morin
Will Porter
Avi Rappoport
Cal Simone
Steve Sisak

TABLE OF CONTENTS

DEPARTMENTS

Soapbox

Itanium, And Pentium, And MacTel, Oh My
by Michael R. Harvey 2

Getting Started

PHP And MySQL, Together At Last
by Dave Mark 8

The Source Hound

Business Unusual: Open-Source With Teeth
by Dean Shavit 18

QuickTime Toolkit

Developing Applications With The QuickTime For Cocoa Kit
by Tim Monroe 36

Tips From Big Nerd Ranch

Life With SQLite
by Aaron Hillegass, Chris Campbell,
and Marquis Logan 52

AppleScript Essentials

User Interface Scripting
by Benjamin S. Waldie 58

Mac In The Shell

launchd: Judge, Jury, Executioner
by Edward Marczak 70

FEATURED ARTICLES



Journaling Bits

A Look At The OS X "Personal Diary"
by Brad Belyeu 14

Indexing Darwin Files

Extending Spotlight's Reach
by Rich Morin 30

Mac OS X Programming

Save Our Screens 101
by David Hill 44

Serving From Home

Running A Mac Web Server With A Dynamic IP Address
by Danny Swarzman 54

Cocoa's New Tree-Based XML Parser

Using NSXML To Parse, Edit, And Write XML
by Jeff LaMarche 62



Soapbox... Continued

players who are going to get bit real hard by this. Developers doing really unique things with the Mac that required nonstandard coding to work will have to find new ways to do that same thing. Stuff written in Assembly is going to need to be reworked, or rethought. If you hand-tuned some of your code, you'll be doing it again. Game developers will run into some issues other segments won't. But, didn't you already do that stuff once or twice before? It's not an insurmountable task. Cocoa developers, you're in pretty good shape. Anyone whose code base is in Xcode is as close to golden as you can get here. For most Cocoa programmers, who weren't making bad assumptions, it's two hours of work, and a couple of check boxes to build the universal binary that will run on PPC and Intel. Carbon developers on Xcode will have some more work to do, ranging from "not too bad" to "I picked the wrong day to stop sniffing glue". CodeWarriors, embrace the horror. You've got a bumpy road ahead, but it's not as though this day wasn't coming anyway with Mertrowerks slow slide to largely supporting only embedded systems. Now, if you've been flogging your ancient code base, doing as little as possible to keep the app running, then it really is time to either step up, and do the job right, or just fade into that long good night.

And, this is important; Apple and Intel are going to be there all the way, providing every bit of support they can to get you guys running on the new hardware. The tools and support coming on line for developers is considerable. Xcode 2.1 was made available on day one. I talked to one small dev who built a universal binary of one of his apps in no time (then all he had to do was wait for the test box from Apple to see if it worked). No one is going to have their cheese left hanging in the wind, unless they decide to hang it out there themselves.

That's not to say there won't be bumps in the road. The best-laid battle plan never survives first contact with the enemy. But, it will ultimately work. The job will get done.

Third, the costs will be out of control. Users, CTO's, and network admins will have to buy everything all over again. No. You won't. This roll out is going to proceed much as the change from the 68000 processor to the PowerPC did in the 1990's, and given Apple's experience thanks to that changeover, it will probably be much smoother than the 68K-PPC transition was. My guess? Well, for example, I'm betting you'll get a .x maintenance release of Adobe CS 2 some time next year that will be a universal binary. You won't see CS 3 until 2007 (CS 2 just came out after all), and it will be a universal binary. The usual upgrade costs will apply. CS 4 in 2009 might be Intel only.

Xplain Corporation Staff

Chief Executive Officer

Neil Ticktin

President

Andrea J. Sniderman

Network Administrator

Joel Torres

Accounting

Marcie Moriarty

Customer Relations

Susan Pomrantz

Board of Advisors

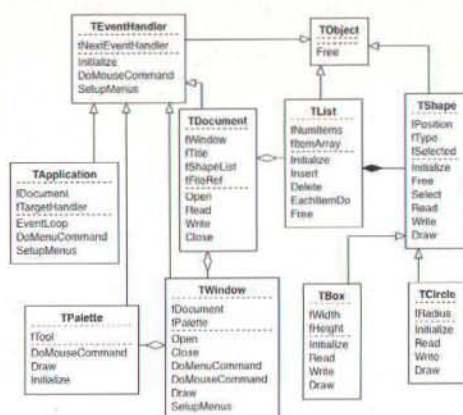
Steven Geller

Alan Carsrud

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are \$47.00 per year. Canadian subscriptions are \$59.00 per year. All other international subscriptions are \$97.00 per year. Domestic source code disk subscriptions are \$77 per year. All international disk subscriptions are \$97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

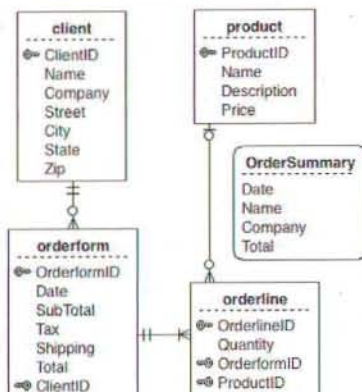
All contents are Copyright 1984-2003 by Xplain Corporation. All rights reserved. MacTech and Developer Depot are registered trademarks of Xplain Corporation. RadGad, Useful Gifts and Gadgets, Xplain, DevDepot, Depot, The Depot, Depot Store, Video Depot, Movie Depot, Palm Depot, Game Depot, Flashlight Depot, Explain It, MacDev-1, THINK Reference, NetProfessional, NetProLive, JavaTech, WebTech, BeTech, LinuxTech, MacTech Central and the MacTutorMan are trademarks or service marks of Xplain Corporation. Sprocket is a registered trademark of eSprocket Corporation. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders.



Code Generation



Data Modeling

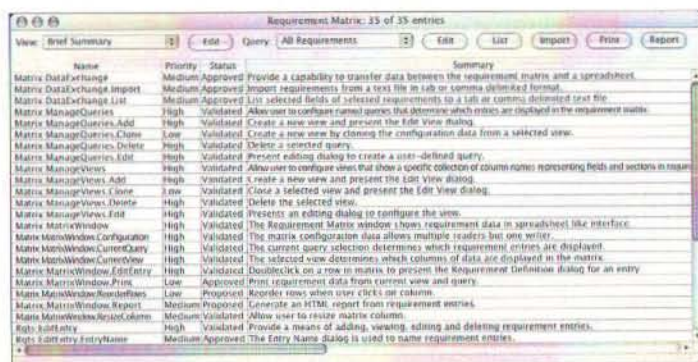


Flexible Reporting

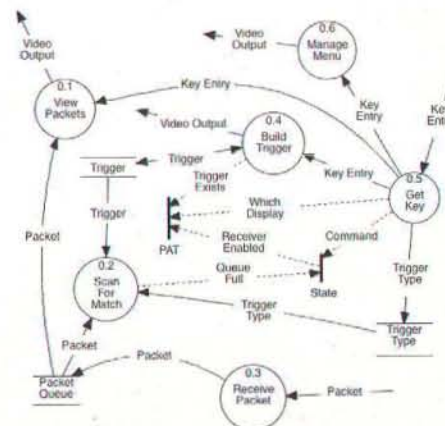
Reengineering

MacA&D OSX

Requirements Management



Structured Analysis & Design



When critical software is needed for the next space mission, automotive control, defense system, telecommunications or desktop application, we can help. MacA&D and WinA&D have been there, done that with 18 years of field proven innovations.

Excel Software provides thousands of developers with tools to design and manage object-oriented software, database systems and embedded, real-time firmware.

www.excelsoftware.com



ES

Excel Software
19 Misty Mesa Ct
Placitas, NM 87043

Ph. 505-771-3719
Fax 505-771-3718

Soapbox... Continued

Everything else will follow similar paths. OS X 10.4.5, say, will ship on the MacTels. It will also work on every PPC Mac that can run 10.4 now. Leopard will be the same, universal binary running on both chips. 10.6 won't be. Apple likes to be out front on that stuff, and pull everyone else along with them. That'll be 2009, probably.

Microsoft, among others, will probably have a somewhat different path. In fact, MS already shared their near term plans at the WWDC keynote. Office 2004 will stay PowerPC only. It doesn't make sense to put the time into a universal binary for 2004 when it will run acceptably under Rosetta, and their next big rev is due around the same time the first MacTels ship. So, universal for Office 2006 (currently identified as Office 12), and maybe 2008, but that could go either way. Versions after that will certainly be Intel only.

There is untold opportunity here, let's not forget. First and foremost, there is a great deal of opportunity for both Apple and Intel. Intel is the 800-pound gorilla. Their processors are everywhere. It's a good idea for Apple to partner with a company with the capacity to meet its needs. Apple brings to the table a great portfolio of IP that can very likely be applied to Intel chip designs. How about Hyper Transport? HT is a big part of what currently gives AMD a leg up on processor power these days. Obviously, AMD was a member of the consortium that developed it. Guess who else was? Apple. Intel just got access to a technology with the ability to help put them up front again, and it didn't cost them a cent in licensing fees.

Now, think on the possibility of quick and easy ports of software that's never been considered for the Mac platform. Had the Navy ran OS X on MacTel, maybe their Aegis cruisers wouldn't have come to a complete halt during testing when the control systems crashed Windows NT. Ponder your accounting department no longer having to worry about the unending deluge of Windows specific viruses and Trojans because your network admin moved all of them to MacTel as soon as PeopleSoft did a build for it. What about the ability to dual boot? Phil Schiller said they won't support it, but neither will Apple actively block it. How long until some enterprising hobbyist programmer has a multi-boot hack up on Sourceforge? One box, and two, three, or four different bootable OSes. I'd be okay with fewer beige boxes under my desk while still being able run all the flavors of Linux, Unix, Windows, and Mac OS I want. Or better yet, how about a version of Virtual PC that runs all that at native speeds, and lets you fast switch between each. Hmm, plenty of opportunity all over the place, it seems.

So, what does it all mean? It means, like Mr. Adams always wrote, Don't Panic. We've got a year or more before the first MacTel hardware will be available for purchase, and even then, all the PPC stuff will still work. It's a long road, with many scenic stops on the way, to an all Intel Mac universe. By the time we all get there, someone will ask, "So, what was all hubbub, bub?"

MACTECH

Communicate With Us

DEPARTMENT E-mails

Orders, Circulation, & Customer Service

cust_service@mactech.com

Press Releases

press_releases@mactech.com

Ad Sales

adsales@mactech.com

Editorial

editorial@mactech.com

Online Support

online@mactech.com

Accounting

accounting@mactech.com

Marketing

marketing@mactech.com

General

info@mactech.com

Web Site

<http://www.mactech.com>

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail**?

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact MacTech Magazine Customer Service at 877-MACTECH

We love to hear from you!

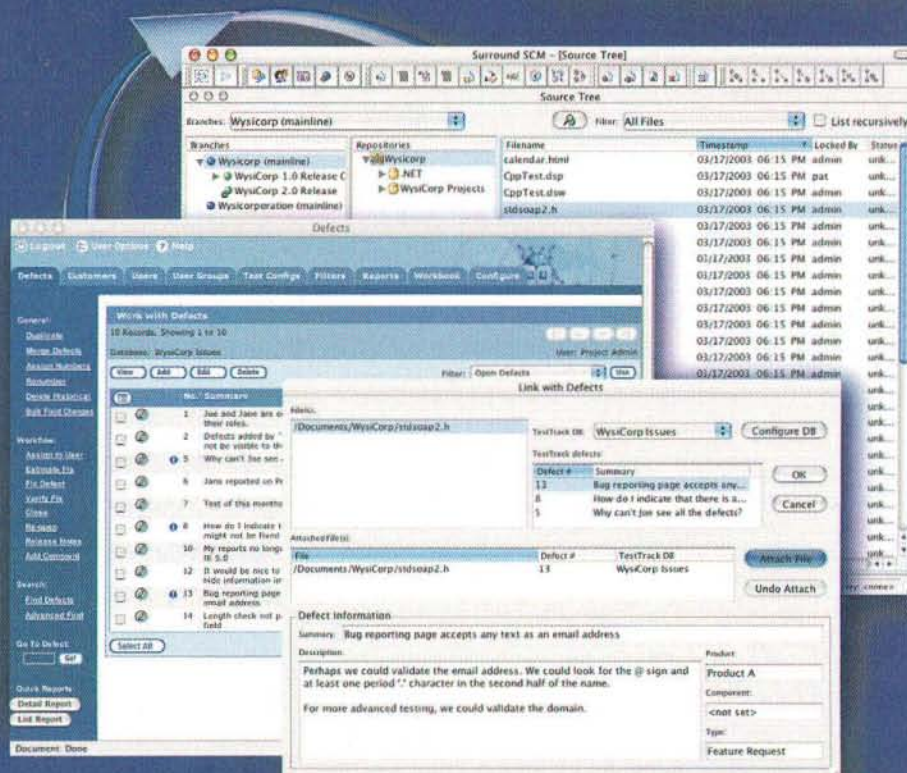
Please feel free to contact us with any suggestions or questions at any time.

Write to letters@mactech.com or editorial@mactech.com as appropriate.

Complete Source Control and Defect Management

Seapine Software™
changing the world
of software development

for Mac OS X



Effective source code control and defect tracking require powerful, flexible, and easy-to-use tools—Surround SCM and TestTrack Pro

- Complete source code control with private workspaces, automatic merging, role-based security, and more
- Comprehensive defect management — track bug reports and change requests, define workflow, customize fields
- Fast and secure remote access to your source files and defects — work from anywhere
- Advanced branching simplifies managing multiple versions of your products
- Link code changes with defects and change requests — know who changed what, when, and why
- Scalable and reliable cross-platform, client/server solutions support Mac OS X, Windows, Linux, and Solaris
- Exchange data using XML and ODBC, extend and automate with SOAP support
- Licenses priced to fit your budget

Seapine Software Product Lifecycle Management
Award winning, easy-to-use software development tools

Seapine
Surround SCM
Seapine
TestTrack PRO



**Download Surround SCM
and TestTrack Pro at
www.seapine.com
or call 1-888-683-6456**

all product names listed herein are registered trademarks of their respective owners. All rights reserved.



PHP AND MySQL, TOGETHER AT LAST

I nterwoven in my last six columns or so were columns that showed you how to install and test PHP and MySQL. The PHP and MySQL columns all dealt with PHP and MySQL in isolation. The PHP code did not access a MySQL database, and the MySQL database access was all done via the Terminal and not via a PHP encrusted web page. In this month's column, we'll verify that both are installed and available, then see if we can't make them play nicely together. Let's start by verifying our install.

Checking Your PHP Install

If you are new to PHP, make your way over to <http://www.php.net> and browse through their documentation. If you've installed even a reasonably recent version of Mac OS X on your machine, you should have PHP installed. One solid clue that you *do* have PHP installed can be obtained by typing this command in Terminal:

```
$ ls /usr/local/
```

If the result of this command includes the text "php5", chances are, you're all set. If it doesn't, visit Mark Liyanage's wonderful PHP site:

<http://www.entropy.ch/software/macosx/php/>

Mark has put together an excellent PHP resource for Mac OS X and he does a nice job keeping it up to date. Though, at this writing, the site still lists Mac OS X 10.3 as

the most recently supported, I had no trouble using his install package on my Tiger machine. Obviously, you'll want to backup your machine first, just in case something does go wrong.

Hopefully, you *do* have PHP installed on your machine already. Regardless of how you got there, once you've got PHP 5 installed, here's a simple test to make sure you're all good.

Launch your favorite text editor, be sure it is set to save as plain-text, create a new file, then enter this text:

```
<?php  
phpinfo()  
?>
```

Save the file as `test.php` and place it in the Sites subfolder of your home folder (inside `~/Sites/`, aka, `/Users/xxxx/Sites/`).

To test your new PHP file, launch Safari and type:
`http://127.0.0.1/~davemark/test.php`

Obviously, you'll replace "davemark" with your own user name. Safari will ask your Apache server to pass the referenced file on to the PHP pre-processor. If all is kosher, a giant table will appear in a Safari window. The beginning of my giant table is shown in Figure 1.

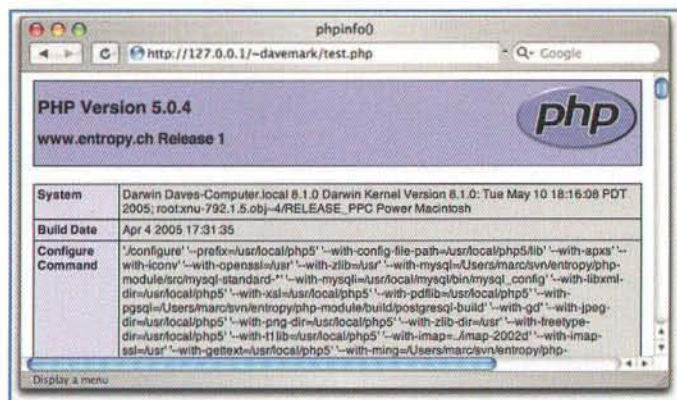


Figure 1. The PHP info table.

Note that my computer is running PHP version 5.0.4. You'll definitely want to check on <http://php.net> to see if there's a later version available.

Checking Your MySQL Install

If you are new to MySQL, check out <http://www.mysql.com>. In the *March, 2005 Getting Started* column, I went through the process of hunting down the proper package for your version of Mac OS X, installing MySQL, starting and shutting down the MySQL server, and setting up the accounts and passwords.

Then, in the *April Getting Started*, we used the MySQL monitor application, running in Terminal, to build a database and, within that, a table. We added to and deleted rows from the table, and ran some queries to report on the table data. Obviously, I don't want to repeat all that here, but I thought it would be worth repeating a few of the basics, just to make sure we were on the same page.

Start by making sure you have an alias set up for mysql. In Terminal, type this command:

```
alias
```

This will list your aliases. Hopefully, one of your aliases will be:

```
alias mysql='/usr/local/mysql/bin/mysql'
```

If not, edit the file *.profile* in your home directory and add this line to the end of it:

```
alias mysql='/usr/local/mysql/bin/mysql'
```

Now quit and re-launch Terminal, which will re-execute the commands in *.profile*. Now, when you type the alias command, your alias should appear. This lets us type mysql to launch the MySQL monitor.

Launch the monitor by typing:

```
mysql -u root -p
```

As a reminder, you are launching the monitor with a user of root. You'll be prompted for a password. Type in the password

you created when you created your account. Remember, this root is not the same root as your Mac root account. If you are having trouble logging in as root, go back to your setup instructions or to the *March Getting Started*.

Here's what my monitor looks like:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.1.12-standard
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql>
```

If mysql does not launch your MySQL monitor, either your alias is not set correctly (in which case, go check to see if you've got a file called mysql in the directory */usr/local/mysql/bin/*) or you did not install MySQL correctly. If MySQL was not installed correctly, get hold of the March MacTech and follow the installation process, or make your way through the MySQL installation instructions on <http://mysql.com>.

Assuming your monitor comes up as mine did, type this command:

```
show databases;
```

Don't forget the ending semicolon. Here's my result:

```
mysql> show databases;
+-----+
| Database |
+-----+
| test     |
+-----+
1 row in set (0.18 sec)
```

```
mysql>
```

If you've played with MySQL, you may find yourself with a different set of databases. Not a problem. As long as the command works, you're fine, no matter the result.

Driving MySQL From Within PHP

In April's column, we used the MySQL monitor to create a database, add rows, delete rows, and update values in a table. In the remainder of this month's column, we're going to do the same sorts of things, but do them from within PHP, instead of from the monitor.

Before we get into our PHP example, let's use the monitor to set up a database and table, then populate the table. This approach is pretty typical: Set up the database and table using the monitor, then query/populate the table from your PHP/web interface.

In the monitor, type this command:

```
mysql> create database products;
Query OK, 1 row affected (0.06 sec)
```


Next, check to make sure the database got created:

```
mysql> show databases;
+-----+
| Database |
+-----+
| mysql    |
| products |
| test     |
+-----+
3 rows in set (0.00 sec)
```

Yup, there it is. Now, let's set **products** as our database and then create a new table:

```
mysql> use products;
Database changed
mysql> show tables;
Empty set (0.00 sec)
```

No tables exist yet. Let's create one:

```
mysql> create table cables(
-> name varchar(60),
-> lengthInCm int(2),
-> cableType ENUM( 'firewire', 'usb' ),
-> cableID int(10) auto_increment primary key );
Query OK, 0 rows affected (0.15 sec)
```

Imagine a database set up for an e-commerce web site, designed to store info on all the site's products. The site sells firewire and usb cables of various lengths. The cables table will store info on the various cables sold by the site. Obviously, this is a *very* simple example. If this were a real database, we'd want to create multiple tables and have them reference each other. For example, we'd probably create a manufacturer table and have a field in the cables table refer to an entry in that table. This table is not very complicated and would not be terrifically useful in the real world, but it *will* serve to demonstrate the connection between PHP and MySQL. Read on...

Now let's populate the table:

```
mysql> insert into cables values ('Varco DX100', 100,
'firewire', 0);
Query OK, 1 row affected (0.13 sec)
```

Note that we embedded the manufacturer's name in the cable name field. Ick. As I mentioned, this should be a reference to a separate, manufacturer table. For now, this'll do.

Notice that we created an enumerated field. The `cableType` field can only take on one of two values, either 'firewire' or 'usb'.

We used a value of 0 for the `cableID`. This asks MySQL to create an index for this entry automatically.

Let's retrieve what we just put in:

```
mysql> select * from cables;
+-----+-----+-----+-----+
| name      | lengthInCm | cableType | cableID |
+-----+-----+-----+-----+
| Varco DX100 | 100         | firewire  | 1       |
+-----+-----+-----+-----+
1 row in set (0.04 sec)
Let's add a few more:
mysql> insert into cables values ('Genenco VT100', 100,
'firewire', 0);
Query OK, 1 row affected (0.38 sec)

mysql> insert into cables values ('Genenco VT50', 50,
'firewire', 0);
```

```
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into cables values ('Genenco U100', 100,
'usb', 0);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into cables values ('Plexicor uShorty', 20,
'usb', 0);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into cables values ('Plexicor fShorty', 20,
'firewire', 0);
Query OK, 1 row affected (0.00 sec)
```

Now let's take a look at what we've got in the table:

```
mysql> select * from cables;
+-----+-----+-----+-----+
| name      | lengthInCm | cableType | cableID |
+-----+-----+-----+-----+
| Varco DX100 | 100         | firewire  | 1       |
| Genenco VT100 | 100         | firewire  | 2       |
| Genenco VT50 | 50          | firewire  | 3       |
| Genenco U100 | 100         | usb       | 4       |
| Plexicor uShorty | 20         | usb       | 5       |
| Plexicor fShorty | 20         | firewire  | 6       |
+-----+-----+-----+-----+
6 rows in set (0.39 sec)
```

Accessing Your Data from PHP

So now we have a database and a table filled with data. Our next step is to access this data and make it appear on a web page.

As a reminder, your PHP statements will be embedded within your html code. Once your *.php* file has been handed off to the PHP pre-processor, the pre-processor will interpret the PHP code and replace the code with the output generated by the code. If this confuses you, here's a very short example from last November's column, just to refresh your memory.

Using your plain-text text editor, create a new file called *mysqltest.php* and save it in your *Sites* folder, right alongside your *test.php* file we created earlier. Enter this code in the file and save:

```
<html>
<head>
  <title>PHP Test</title>
</head>
<body>
  <p>This is some pure HTML loveliness.</p>
  <?php
    echo "<p>Hello, World!</p>\n";
  ?>
  <p>Did we echo properly?</p>
  <?php
    echo date("r");
    echo "\n";
  ?>
  <p>It works!!!</p>
</body>
</html>
```

Once your file is saved, go into Safari and enter this address:

<http://127.0.0.1/~davemark/mysqltest.php>

Make sure you replace "davemark" with your own user name. Figure 2 shows what I saw when my page loaded. The first line was produced by the HTML. The second line was the result of the output of the PHP *echo* function. The *echo* function produced some HTML which was added to the stream. Next

came another line of HTML ("Did we echo properly?"), followed by another pair of echos, echoing a date string and a return. This is all polished off by a last line of HTML, generating the string "It works!!!".

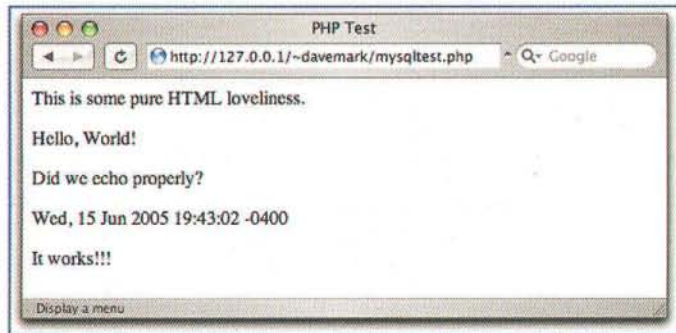


Figure 2. A simple PHP test.

If you view source on this output, here's what you get:

```
<html>
<head>
  <title>PHP Test</title>
</head>
<body>
  <p>This is some pure HTML loveliness.</p>
  <p>Hello, World!</p>
  <p>Did we echo properly?</p>
  Wed, 15 Jun 2005 19:43:02 -0400
  <p>It works!!!</p>
</body>
</html>
```

Go back and look at the original PHP. Make sure you understand how the original PHP got translated into this source. Remember, every chunk of PHP code in the original source was replaced by its output to achieve this HTML listing.

Connecting to the Database

OK, now we're ready to fetch our data. Our first step is to connect to our database. Open *mysqltest.php* and replace the contents with this code:

```
<html>
<head>
  <title>MySQL Test</title>
</head>
<body>
  <p>Connecting to the database...</p>
  <?php
    $host = 'localhost';
    $user = 'root';
    $pw = '';
    $db = 'products';

    $link = mysql_connect( $host, $user, $pw )
      or die( 'Could not connect: ' . mysql_error() );
    echo 'Connected successfully';
    mysql_select_db( $db );
  ?>
  <p>If we got here, we've connected!</p>
</body>
</html>
```

In the above code, replace the *\$user* and *\$pw* string values with whatever user and password you used to create the database. Save the code and reload this page in Safari:

<http://127.0.0.1/~davemark/mysqltest.php>. Figure 3 shows my results.

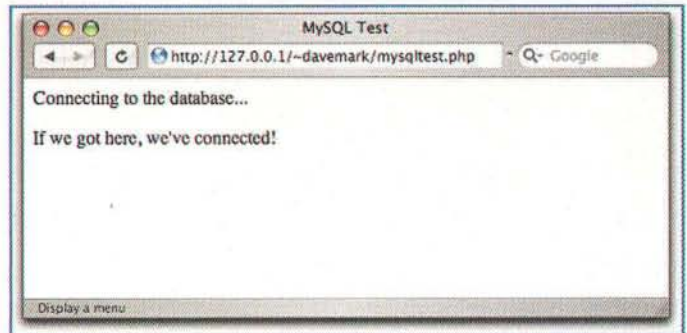


Figure 3. Connecting to the database.

In a nutshell, we used the function *mysql_connect()* to connect to the database and the function *mysql_select_db()* to select the database. This is like logging in using the MySQL monitor, then saying *use products*.

One line of code worth taking a second look at is this one:

```
$link = mysql_connect( $host, $user, $pw )
  or die( 'Could not connect: ' . mysql_error() );
```

Notice the use of "or" here. This is a pretty common technique in PHP. The second part of the or clause will only execute if the first part fails. The *die()* function is equivalent to *exit()*. *die()* will post the passed in string as output, then exit.

Notice the use of the "." operator to concatenate two strings together. This is another common PHP technique. In this case, the "." operator will build a single string from 'Could not connect: ' and the string returned by *mysql_error()*. *mysql_error()* returns the error message from the previous MySQL operation.

Our next step is to query the database and to print the data we retrieved.

Querying the Database

Back in your text editor, open the file *mysqltest.php* and replace its contents with this:

```
<html>
<head>
  <title>MySQL Test</title>
</head>
<body>
  <p>Connecting to the database...</p>
  <?php
    $host = 'localhost';
    $user = 'root';
    $pw = '';
    $db = 'products';

    $link = mysql_connect( $host, $user, $pw )
      or die( 'Could not connect: ' . mysql_error() );
    echo 'Connected successfully';
    mysql_select_db( $db );
  ?>
  <p>Here's the table data:</p>
  <?php
    $sql_statement = "SELECT * FROM cables";
    $results = mysql_query( $sql_statement )
      or printf( "Query error: %s", mysql_error() );

    while ( $row = mysql_fetch_assoc( $results ) )
    {
```



```

        echo $row['cableID'] . ": ";
        echo "\" . $row['name'] . "\" . ", ";
        echo $row['lengthInCm'] . ", ";
        echo $row['cableType'];
        echo "<p>";
    }

    mysql_close( $link );
    ?>
</body>
</html>

```

Save the file and reload it from Safari. Figure 4 shows my version of this run. Notice that the data is lightly formatted, with a colon (":") after the item number and commas between each of the fields.

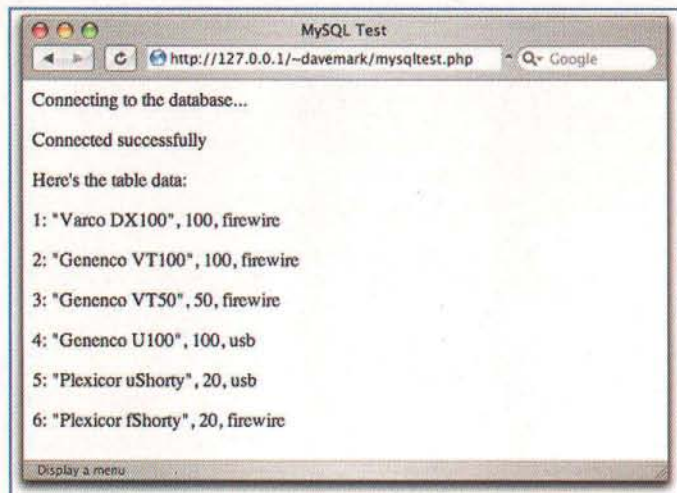


Figure 4. Retrieving the data from the database.

Let's take a look at the code. We started with the original code, connecting to and selecting the database.

```

<html>
<head>
<title>MySQL Test</title>
</head>
<body>
<p>Connecting to the database...</p>
<?php
    $host = 'localhost';
    $user = 'root';
    $pw = '';
    $db = 'products';

    $link = mysql_connect( $host, $user, $pw )
        or die( 'Could not connect: ' . mysql_error() );
    echo 'Connected successfully';
    mysql_select_db( $db );
    ?>

```

Next, we spit out a line of HTML, preparing us for the table data to follow.

```
<p>Here's the table data:</p>
```

We load our query into a PHP string, then pass the string into `mysql_query()`. This is the equivalent of typing the string into the MySQL monitor as a query. Once again, we use the "or" operator and this time, instead of exiting with `die()`, we'll display an error message using `printf()`. `printf()` is derived from its C forbearer, but is a part of the PHP library. As a point of interest, `echo` is *not* a

function, but is a language construct. You can use `echo` as a statement, but can't pass it as a function. Use `printf()` instead.

```

<?php
    $sql_statement = "SELECT * FROM cables";
    $results = mysql_query( $sql_statement )
        or printf( "Query error: %s", mysql_error() );

```

Next, we enter a while loop, using `mysql_fetch_assoc()` to fetch one row of the table at a time. `mysql_fetch_assoc()` returns an associative array, which is an array indexed by name instead of by number. Instead of `$row[3]`, you'd refer to `$row['cableType']`. Associative arrays are one of my favorite parts of PHP. Note that you could have also used `mysql_fetch_row()`, which would have returned a more traditional, numerically indexed array.

```
while ( $row = mysql_fetch_assoc( $results ) )
```

For each row of data, we echo the field value, interspersed with colons, spaces and commas. Note that we also make frequent use of the "." operator.

```

        echo $row['cableID'] . ": ";
        echo "\" . $row['name'] . "\" . ", ";
        echo $row['lengthInCm'] . ", ";
        echo $row['cableType'];
        echo "<p>";
    }

```

Finally, we close the database using the value returned by `mysql_connect()`, then exit our PHP area and return to HTML.

```

mysql_close( $link );
    ?>
</body>
</html>

```

Until Next Month...

Once again, seems we just get started when I've run out of room to write. <sigh>. Your assignment for this month is to first do a bit of research, then take the result of this month's query and build a nice HTML table instead of just dumping the data using `echo`. On the research side of things, go to <http://php.net> and dig down through the PHP manual, looking up the various functions we played with in this month's code. For example, here's a link to the page that talks about `mysql_fetch_assoc()`:

<http://php.net/manual/en/function.mysql-fetch-assoc.php>

Take some time to get to know the PHP documentation. You'll find it chock full of examples and incredibly useful. Enjoy! ☺

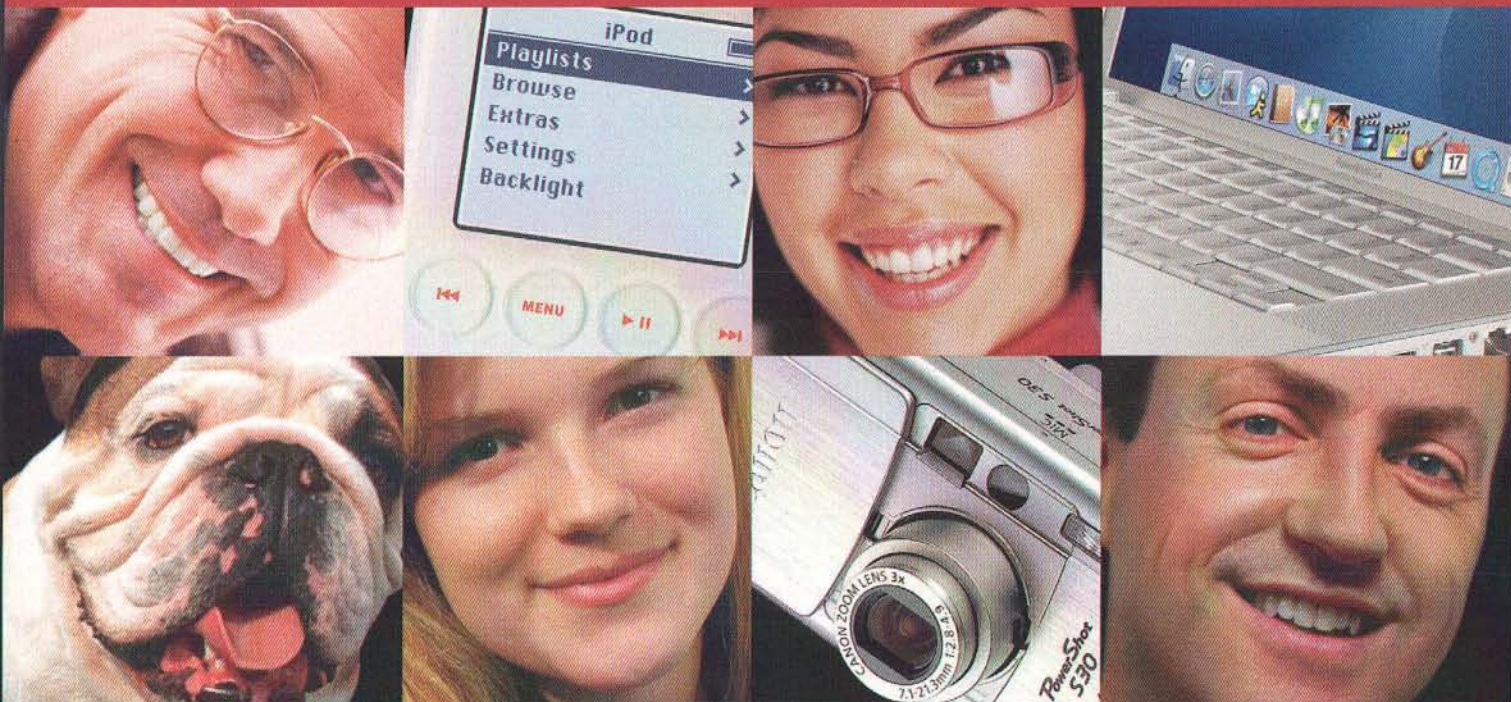


About The Author



Dave Mark is a long-time Mac developer and author and has written a number of books on Macintosh development. Dave has been writing for MacTech since its birth! Be sure to check out the new Learn C on the Macintosh, Mac OS X Edition at <http://www.spiderworks.com>.

At Small Dog Electronics, happy customers are our highest priority.



When you shop at Small Dog Electronics, you get more than just great selection and low prices; you also get personalized service from genuine Apple Professionals who take customer service very seriously. And that's a promise... no if's, and's or but's.

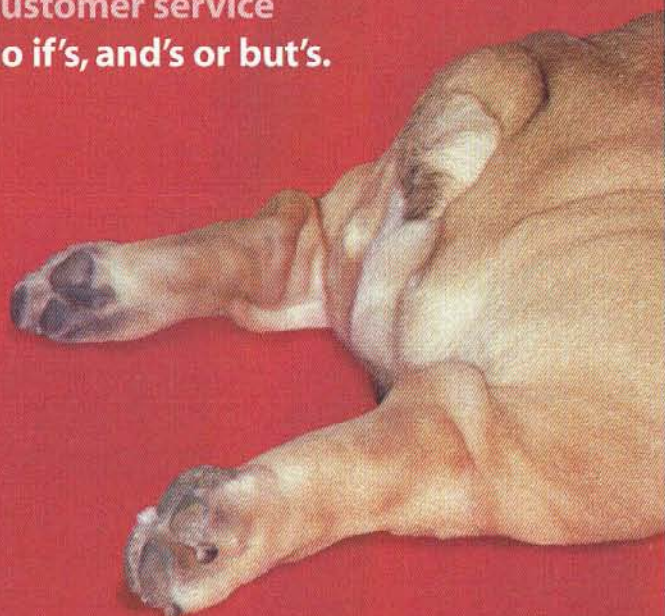


**Small Dog
Electronics**

www.smalldog.com



Apple Specialist



1-800-511-MACS

A socially responsible business since 1996

Journaling Bits

A Look At The OS X "Personal Diary"

By Brad Belyeu

Welcome

My typical work day is filled with interruptions. Phones constantly ringing; people asking questions, etc. Everyone wants a piece of my time. Interruptions are not only a major hit to productivity and effective time management; but when you're trying to replace a PowerBook logic board, it can mean bigger problems. They sometimes leave me wondering, "Where did I set my screwdriver?", or "Where does this screw go again?" Now imagine having a secretary that follows you around all day recording what you do. Whenever you finish a task, it is scratched off a list; and whenever you start a new task, it is added to a list. Anytime you had an interruption, you could simply ask your secretary what you were doing before the interruption. That would be amazing! This is what Apple's software engineer's were thinking when they introduced file system journaling.

Meet The Secretary

To understand why file system journaling is important, we need to understand file systems & take a look at OS X systems prior to journaling. File systems are like library cataloging systems. Library catalogs can tell you an exact name & location of any book in the library; just like file systems can tell you the exact name & location of a piece of data on a drive. Both systems regulate naming & location conventions for what they store. If a file system gets corrupted, it would be like trying to read a library catalog in a language you can't understand.

Journaling was introduced into OS 10.2.2 Server edition and into the client in 10.3. In previous OS versions, when you have any of these failures, they leave your system volume in an unknown state because there is no record of where the disk was interrupted. The OS then had to do a data integrity check of the entire disk. Interruptions can also damage system files rendering the boot volume unusable. This is usually when you restart and are presented with a folder and the infamous flashing

question mark because the computer can't find a disk to startup. But when journaling is enabled, the computer tracks file system operations and immediately logs them in a journal. When restarting after a failure, the operating system can use the journal to return the files system to its previous state. This eliminates the need to check the entire drive for inconsistencies. Instead, when the computer is restarted, it takes a look at its journal, reads to find out what it was doing, and picks up right where it left off. File system journaling is a major advancement in protecting the file system from power outages, hardware failures, or software crashes.

The Good

A system continuity failure interrupts the read & write processes of a hard drive. These interruptions create discrepancies between the file system (library catalog) and the actual location of the stored files (location of books in library). In a journaled file system the OS has a record of disk activity, preventing the disk from becoming unusable (most of the time) which also prevents repairs that cost both time & money. It also cuts down on the time it takes to restart the computer after one of these failures because the entire disk doesn't have to be checked for inconsistencies block-by-block. Although a multi-gigabyte hard drive only takes several minutes to check, the original idea in introducing file system journaling was to minimize down-time for servers because a multi-terabyte disk could take hours to check. A journaled file system takes just a few seconds to return to operating condition no matter what the size of the drive is.

File system journaling was developed by Apple to be an extension to the current file system. This is good news because it lets you enable/disable journaling without reformatting the disk! The Mac OS Extended volume format (HFS+) can change between journaled

and non-journaled without any data loss. The advantages of this will become clear later in the article as we discuss reasons you may not want to use file system journaling.

Because the journal works at the system level, it is invisible to the applications on the drive. Other than some disk utilities, all applications and network setups are compatible with journaling. Most 3rd party disk utilities will also work with journaling. Just in case you were wondering, finished actions are removed from the journal so the journal file doesn't become too large. Just like that the secretary crosses completed tasks off your list and throws them in the trash.

Journaled file systems are backward compatible with the regular HFS+ file systems. Meaning that computers using an older Mac OS can have full access to journaled volumes. This is great news for external hard drive users with multiple computers. That way you can use journaling on your hard drive and all your Macs can use the drive whether they are running OS 10.3 or an earlier version.

The Not So Good

Journaling doesn't come without its cost, which happens to be performance. But fortunately, the performance hit only comes when the system is writing to the disk and not when it is reading from the disk. I've never been able to actually tell the difference in speed on drives running journaled verses non-journaled drives. Supposedly, if you have a client version of the OS, the performance hit can be up to 20%, but the server version of the OS comes with a buffered version of journaling. If you have enough RAM, the server version is only supposed to take a 2-3% hit on hard drive performance.

Here's my general opinion on two occasions when you shouldn't use journaling. If you have a drive that is primarily set up to be read-only access, you don't need journaling. For example, if you have a hard drive that you keep your applications on (or just the OS), but not your actual documents or working files on, then you probably don't need file system journaling. This is because you are just pulling data off the drive 99% of the time and are rarely saving data back to the drive. If your system fails, your drive probably wasn't in the middle of writing data. Or if you have a file server disk that is always under heavy speed demands AND you back it up on at least a daily basis, I would consider disabling file system journaling. This might be the case if you are storing large data files containing audio or video that are accessed frequently.

Although file system journaling will help you save data that has already been journaled to your drive; you may still lose data in the event of a failure that is in the buffer before it is written to the drive. So file system

journaling definitely needs to be complimented by a battery backup system.

The last drawback to consider is the failure of some 3rd party disk utilities to work with file system journaling. Make sure you check compatibility on a manufacturer's website before using a 3rd party utility on a journaled file system. Many of the vendors released updates specifically for their product to work with a journaled file system. I've heard nightmares of people losing all their data by running an incompatible disk utility with a journaled file system.

File system journaling has its limitations and is not a fix-all answer to your data problems. According to Apple, file system journaling is one of several measures you should take to avoid data loss. File system journaling should be coupled with RAID (redundant array of indexed disks) storage, a continuous backup strategy, and an uninterrupted power source (UPS) for maximum protection of your digital assets.

Enabling/Disabling

You can enable or disable file system journaling using Disk Utility (located in the Applications/Utilities folder). Select a volume in Disk Utility (figure 1).

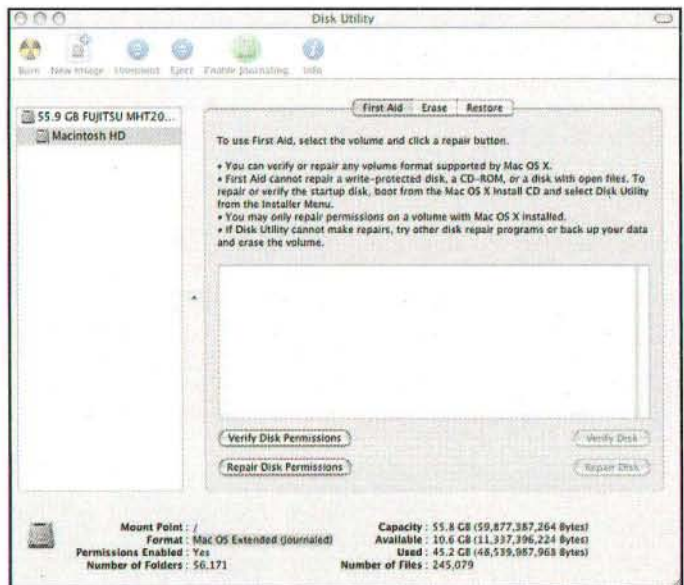


Figure 1.

If you want to enable journaling, you can simply click 'Enable Journaling' from the pane. This will enable journaling without erasing data on the disk. If you'd like to disable journaling, you can select the volume and then click File-Disable Journaling from the menu bar.

You can also enable or disable file system journaling using the command line version of Disk Utility, diskutil. To have sufficient privileges you'll have to use sudo (super user do) with diskutil. To

enable file system journaling on the root directory type:

```
sudo diskutil enableJournal /
```

and then type in your administrator password. If you'd like to disable file system journaling on the root directory use:

```
sudo diskutil disableJournal /
```

followed by your administrator password. To enable or disable on volumes other than the startup disk replace / with /Volumes/(VolumeName).

Repairing Journaled Drives

To repair an ailing journaled drive you can once again use Disk Utility. The trick is that you can't repair the disk you booted from; so if you want to repair your startup volume with Disk Utility, you have to startup to a CD or another drive. Once you have Disk Utility open, select the drive that you want to repair and then click 'Repair Disk'. If you don't have another drive you can startup from you'll be better off using a 3rd party utility or the command line.

If you don't want to spend the money on a 3rd party disk utility, try using the command line utility fsck_hfs. To force fsck_hfs to check a journaled

volume you need to use the argument -f. Also, if you just want to verify status of the drive, use the -n flag; but if you want to repair the drive, use the -y flag.

There are a number of excellent 3rd party disk utilities. My favorite are DiskWarrior X & TechTool Pro. There are also a number of shareware & freeware applications to be found just by searching versiontracker.com or macupdate.com.

Wrapping up Journaling

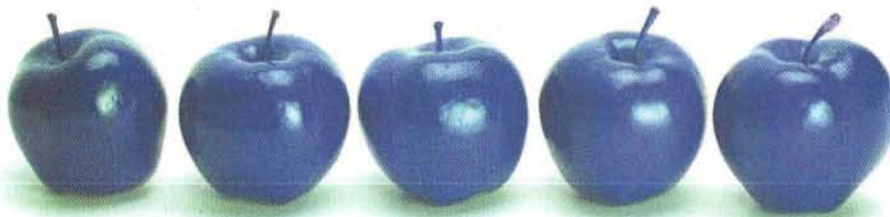
Using file system journaling is a great idea for most users. There are the occasions to disable it, but for the most part it makes our lives easier. That is where the analogy leaves the library catalog. File system journaling is done automatically, but looking up a book in a library can sometimes be challenging. I'm just glad file system journaling keeps track of everything without my lifting a finger. But remember, file system journaling does not excuse you from needing a regular system backup!

MI

About The Author

Brad Belyeu is the President of ABCConsulting based out of Oklahoma City, OK. He is an Apple Certified Technician and a certified member of the Apple Consultant Network. Most importantly, he is happily married.

Mac OS X



PORTLOCK™
portlock.com

Apple®, Linux®, NetWare®
and Windows® Platforms

101 North Main Street • Butte, Montana 59701 • Ph: 406-723-5200 • Fax: 406-723-5205

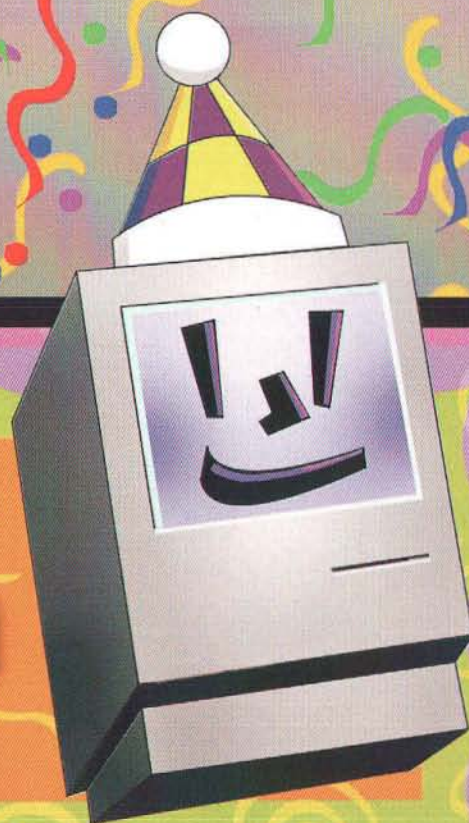
© 2004 Portlock Software. All rights reserved.

Apple is a registered trademark of Apple Computer, Inc., Linux is a registered trademark of Linus Torvalds, Netware is a registered trademark of Novell, Inc., and Windows is a registered trademark of Microsoft Corporation in the United States and other countries.



This is Your Computer

This is
Your Computer
With Better Ram



better **RAM**.comsm

it's better here

Memory built to order - so you always get the right memory with up to the minute prices!

Toll Free: (800) 895-3493 Outside US/Canada: 805-494-9797 Fax: 805-494-9798

BUSINESS UNUSUAL: OPEN-SOURCE WITH TEETH

Dear reader, if you've been reading the Source Hound, you'll know that I'm a big proponent of Apple's use of Open Source in OS X as well as using Open Source software to enhance the OS X experience. You might think that I'm a bit unusual in this business, but I do have a kindred spirit whose passion for Open-Source has taken him out of the comfortable confines of a writer's office, to directly into the line of fire. The person I'm speaking of is Michael Bell, a.k.a Drunkenbatman, (we'll call him DBM for short), proprietor of Drunkenblog (<http://www.drunkenblog.com>), a web site which has raised more than a few eyebrows by its rigorous defense of those targeted by Apple in the Tiger leak lawsuits, in which even Steve Wozniak weighed in with an email to Drunkenbatman as well as 25 other major Mac developers.

Editor's Note: The following discussion touches on many controversial and potentially sensitive areas. We want to make it clear to our readers that, while we find this discussion compelling, enlightening and important, the opinions expressed herein are those of Michael Bell and Dean Shavit, and do not necessarily reflect the opinions of MacTech or its parent, Xplain Corporation.

Taking a Bite out of Open-Source

Most recently, DBM has spent hundreds of hours deconstructing a software company called Maui XStream, and their CherryOS product. His investigation into Maui XStream is both interesting and technically enlightening, I encourage MacTech readers to check it out. As you will see, DBM has declined to talk directly about the CherryOS/Maui XStream situation.

I, the Source Hound (TSH) had a virtual sitdown with DBM (over iChat) to discuss Open-Source software (OSS) intellectual property (IP), and other issues as they relate to OS X and OSS in general. While we agreed on some points, we diverged on others. One point we both agreed on is that the world in general and OS X needs OSS, and that OSS transcends the interest of any one company, person, or platform.

Drunkenbatman in the House

DBM: drunkenbatman in the house.

TSH: Just a little background to get things going.

DBM: is this going to be verbatim or cleaned up?

TSH: I'll clean it up with your approval.

DBM: w00t

TSH: I've been reading your blog "Drunkenblog" <http://www.drunkenblog.com> for a couple of years now

And I've been amazed at the steady progression of effort that's been going into your articles.

DBM: Um, are we doing the interview right now? Lol you have to say *go*.

TSH: OK, let me get us warmed up, *go!* How did you get started with Drunkenblog, why Drunkenbatman as an identity, what about the cow?



Figure 1. The Drunkenblog Cow

DBM: Erm, its a little weird to me that I've actually been doing it for a few years. There's kind of a cut-off point between when people were actually reading, and when I actually had it up. I'd have to check, but I think the first post is "So, I'm doing this because everyone is and you're supposed to, but don't really know what to say" and then a month of nothing. And then a terminal tutorial or two, and then things started to go bang. Back then, it was originally DrunkenBatman.com, because I used the nickname 'drunkenbatman' on a lot of places I hung out. 'My Blog' sounded lame, and so did 'Drunkenbatman's Blog'. Then I got a weird letter from Time Warner, and while they eventually

backed down, I decided it was good to pick up DrunkenBlog.com and here we are. The Cow is just there because it cracks me up. Are you asking if I actually drink while I write?

TSH: No I don't think you drink when you write.

DBM: But I *do* sometimes. If you check the time stamps, I'm often posting at 3am or 5am, so you often need things to keep you awake or things to settle you down. You can generally tell when I've rounded the horn with the rum during the making of a post, as typos go way up.

TSH: I know how hard it is to write interesting, well-researched technical articles, so I am sure you must be sober the majority of the time. So run-ins with lawyers are nothing new for Drunkenbatman?

DBM: They're pretty new, and the thing with Time Warner wasn't really a big deal. They just wanted the domain, and I had to explain that for a variety of reasons that wasn't an option, and they agreed to walk away as long as I wasn't using it for anything commercial. I can kind of get where they're coming from — they might not care that much about what I'm doing personally, but if it's their trademark and all, they have to defend it or they can have problems when someone starts hurting the image of their "hero."

TSH: You had mentioned a while ago that you came from the user interface side of the computer business, but for some reason, reading Drunkenblog convinced me that you were a developer.

DBM: Yeah, that is a misconception I get a lot, which is actually kind of cool. I can certainly code some, but no I wouldn't call myself a developer.

I think a lot of that misconception for people is rooted in the fact that I know a lot of technical things, and about the developer side of the process of creating software. And when you're making an application, there are certainly sides, even though someone might have to be wearing multiple hats.

TSH: You know a whole lot about OSS (Open-Source Software), as much as anyone I've spoken to. How would you characterize your role in the OSS great chain of being?

DBM: God.

Or a minnow. Sorta depends on the day.

Lol sorry. Um, can I ask why you're asking?

TSH: I want you to tell me if you're a hobbyist, journalist, aficionado, spy, etc. How do you characterize yourself?

DBM: Honestly? Reluctant poster boy for a whole slew of causes. When it came time to do some interviews, I ended up deciding that a lot of the commercial guys are good at getting their name out, and I wanted to pick people who were slaving away—they don't generally ask for money, but maybe I could shine some light on the kick ass work they're doing.

TSH: You know that's exactly what the Source Hound Column is all about, bringing attention to Open-Source projects! However, you're the first interview I've ever done!

DBM: So. . . CherryOS came about because Pieter Van den Abeele, the head of gentoo-alt (they do gentoo macos, gentoo for powerpc, etc.) forwarded it onto me.

TSH: The amount of effort you put into researching CherryOS was astonishing.

DBM: It was an off-the-cuff email from one of the developers where he asked "what's this about you getting ripped off", and he just thought I could maybe do something to help, since I'm known for putting together some big things

TSH: Big things such as. . .

DBM: I had Steve Wozniak give me a quote for the site and twenty-five plus other big-time Mac developers. None of that happened by accident, I went to all of them and asked. I knew Woz from ages ago in a minor way; then I took his quote and went to the others.

TSH: Your technical knowledge seems eclectic and granular, you sprinkle your articles with it, adding flavor. Where does this knowledge come from? Is it part of your day-to-day work environment? I was a big Mac fan even before the OS X days, how about you? Are you willing to date yourself? I am!

DBM: Lol, that's a lot of questions at once. I know a lot of things I don't say, but a lot of it just comes just because it's something I happened to be interested in and went nuts researching. And really, when it comes to technology, it's like Mark Twain said: "History doesn't repeat itself, but it rhymes." There are patterns in most of what we use, and once you grok the patterns, you can have an idea of how things will or could go. For example, a lot of the technology we use right now is really an extension of stuff that was around in the 70s and 80s. You pull someone here with a time machine, and they'll understand the algorithms and equations being used, even if they didn't have the horsepower to use it as pervasively as we do. If you grok one protocol, you usually have the patterns in the rest. You grok one object oriented programming language, and you know the patterns of what's up the others, the rest is just implementation.

TSH: Of course, it's the same with scripting languages. A loop's a loop; if you've seen one subroutine you've seen them all. . .but I'm more interested in finding out how you came to write about OS X and the Mac. . .

DBM: As for dating myself, my first Mac that I actually owned was a Quadra 660 AV. I loved that machine, and lost approximately one man-year to playing Bungee's Marathon. I loved that machine. It was so starkly different from anything else on the market at the time that the only thing I had to lust for were some of the machines SGI was putting out, or its big brother, the 840AV. I'll never forgive Apple for the GeoPort, but I loved that machine.

TSH: Yeah, I had a Geoport telecom adapter too, it wasn't one of Apple's best efforts. . .

DBM: I started using Macs because our high school used them for the school newspaper, and I was really interested in the arts at the time. I had a really supportive family, and my dad

basically used his earnings from his summer job to pay for it, which was a really big deal. If I recall, it was about \$3,500 with the monitor, plus the GeoPort adapter that shipped over 6 months later, which was a huge chunk of cash to lay out for a kid in his bedroom.

TSH: OK, so now having dated yourself, I'm going to hit you with this question. . . Do you remember COS?

DBM: I'm not the best with acronyms. COS? CherryOS?

TSH: Not CherryOS, but a predecessor, both in opacity and form. . . CryptoOS.

DBM: Lol, no, I honestly can't say that I do, although the name sounds familiar. Emulators in those days were hardware cards you'd slap into your computer; even Apple was selling one. Was someone trying to do it all in software then?

TSH: Not exactly. It was a hoax perpetrated by some folks in Germany, who claimed they'd written a new OS for the Mac that was encrypted throughout. This was when we were waiting for Copland, 1996 or so.

DBM: *Oh yes*, I do remember them.

TSH: I remember they kept it up for quite a while, until one day, they actually *asked* on their website for a copy of Copland. They soon disappeared after that.

DBM: They never actually produced anything—it was all just a press release here and there with a simple website, right?

TSH: Yup, they generated *a lot* of buzz. We were so desperate for something "new" for our Macs, that we would've taken just about anything that wasn't System 7.5! I couldn't help but make some sort of strange connection between COS of the 1990s and the COS of 2004 and 2005!

DBM: *grins* I actually think hoaxes are kind of cool for just that reason. They can show what we want to believe. The fine line between a hoax and a fraud is when they're asking for your cash.

TSH: Ah, that is the line, isn't it . . .so it seems our Mac lives intersect somewhat, though I'll have to admit that the computer I used in High School was an Apple II, and it was brand-new.

DBM: Hey, I used an Apple II in school too, because our schools weren't really rolling in it and once you have Number Munchers you're really just adding nicer graphics. The nicest computers were in the media lab and newspaper, which were primarily LC II's and III's and one Quadra. Oregon Trail, all the way.

TSH: Do you work in a Mac-related field now?

DBM: No, I don't really work in a Mac-related field, I just try to work in fields where what I'm using doesn't matter.

TSH: That's really easier said than done, so I'm guessing that you must be a writer or wordsmith of some sort.

DBM: In some ways, it gets harder and harder as time goes on, even if it's not a fun subject to talk about. I have friends who do nothing but web work, which you'd think would just require a text editor and web browser, and at worst VirtualPC in order to test stuff in IE for Windows. However, it's all those little vertical apps that just kill you. You'll be consulting for someone, and they use Source Safe to version their code, which has no Mac client to speak of, so you're just screwed. I guess what I'm trying to say is that what I do for a living isn't that important.

TSH: So some of your passion for Open-Source Software is work-related? I know mine is an outgrowth of wanting to avoid the verticals you mentioned.

DBM: A lot my passion about OSS comes from using and working with the apps, and I was originally fascinated by the idea of giving your work away. When you consult for someone, I think your job is to do what's best for the client, even if that means you end up picking up a PC. I don't have any problem with people using Windows or locking themselves in, even if it annoys me.

TSH: Interesting. "Giving it away" versus "sharing." There's a big difference there.

DBM: Yes, there is, but when I first got into it, like most, I didn't really get what was going on. All I knew was you could download this stuff, and the source, and use it for free. Peripherally, I knew there were things going on with MySQL and KDE that were causing a stir, but I wasn't really aware. And that's probably most people's exposure to Open Source. They'll be using something proprietary, and something comes along that's 'free' so they download it and use it. Sometimes what's 'free' is actually better than what's proprietary. And then they're not making excuses for what they're using to save a buck, so they become loyal. After a time, they might even become a believer. My passion for it now is primarily for practical reasons. We need software. Lots of it, piles of it. More people are being born than are going into programming, and while new coding paradigms and tools can act as force multipliers, we need to improve what we have and create new things. More and more often, those new things are being based on OSS, because reinventing the wheel just gets old.

TSH: In my opinion, free is always better than proprietary. It makes me a little ill to see software being sold (even in the OS X domain) that isn't as good as the OSS alternative. Yet many find a comfort in paying for it, even if they know they shouldn't. I'll agree that we're going to need more software, but we also need processes and quality control and testing and new means of version control and interoperability. The wheel is really all we have right now, it's often about the hubcaps and how shiny they are. . .

DBM: *grins* Many people have that opinion, but there's a difference between proprietary and free. I don't have a problem with closed-source software, but I get entirely annoyed at software that locks you in. Our data is our life, and I've lost count of the apps that I've bought that died and all that data was locked away within them, because I couldn't upgrade the app when there was a new OS or something. And yes QA (Quality Assurance) is certainly in real trouble, and yes much of it comes from companies having to meet the requirements for the quarter, but those types of things come up in OSS too.

For example, pressures are just often more indirect. The more users you have, the more things they want. If you choose not to put them in, you run the danger of your app being forked. Hell, you just may need to get something out because you'll be going on vacation. One OSS app I know rushed a release a bit more than they probably should have, just because they wanted to make the deadline for Apple's awards at WWDC.

But really, much of the great software I use is 'closed source',

but none of it is proprietary. Most of the great RSS readers are closed source, but they can all import and export your data.

TSH: OK, we have slightly different definitions of proprietary.

DBM: Probably. But hey, why do you use a Mac? It's proprietary, yes?

You could pick up an X86 and run Linux and know everything there, with the exception of perhaps a video driver, is Open Source.

TSH: To me, proprietary means "closed source" and not necessarily proprietary in the sense that the data can't migrate out of the app. I've been over that ground before in my columns.

DBM: Ah, right. You're talking about formats primarily then.

TSH: Basically, I *do* have quibbles with some of Apple's proprietary practices, especially in the area of QuickTime and Fairplay. Also in the area of hardware that's unsupported by OS upgrades.

DBM: And their ROM?

TSH: The ROM is a sword that cuts both ways. At least there's less and less of it now. Man, who's wagging who here?

DBM: Yeah, some of those situations are a little uncool. Apple and what they do and don't support isn't helping their warm and fuzzy image. Wagging is fun. Share.

TSH: Well, let's just say that I'm on the record as supporting OSS on OS X and Apple Hardware for several reasons. First, while I am comfortable with Linux, I prefer Apple Hardware and the consistent GUI that OS X offers. Like yourself, I grew up on Apple hardware and the Mac, then made it my professional choice. I think OSS on OS X offers the best of both worlds, a cliché, I know, but it's really true. And the degree to which they protect certain things is really silly, especially when it comes to the lawsuits over the Tiger "leaks" you wrote about. . . of course, being a developer, I had Tiger, but didn't see what all of the fuss was about. In some respects, I think a lot of it is smoke-and-mirrors. Today, I'm using Tiger and it's great, don't get me wrong, a real advance in many ways, but like you said, some of the QA has gone out the window and it makes me wonder if the lawsuits were just a way for Apple to let the world know "we've got something really good, and it's a secret."

DBM: 10.4 probably never should have shipped in the state it was in, and most people know it. The fact that some real hardcore Mac people are making some rumbles is telling, as it's starting to freak some people out. The truth is, 10.3 never should have shipped in the state it was in either. Almost everyone who was a developer was saying "This is really going GM? My god, this isn't ready," yet ship it did. And then it blasted data off of people's Firewire drives, and FileVault is still one of the scariest and flakiest technologies around.

TSH: FileVault is actually pretty simple. But I know what you mean. Well, quality assurance issues aside, it was the "secrecy" surrounding Tiger that irked me somewhat. I mean, I'm a Select ADC member, I've paid a fee, and when it went GM, I couldn't download the GM build until the night Tiger was released. There were some real surprises waiting for me too! Anyway, we're getting off topic.

I wanted to explore proprietary v. OSS.

DBM: Understood. So, let me swing back around to why I love OSS. Mac OS X really shows off exactly why I've fallen in love with OSS, as there's an old expression "Standing on the shoulders of giants." I doubt we'll see another major kernel rewritten in our lifetimes. Apple proved they couldn't even really do it, at least not with the current management and requirements they had. Copeland didn't just die, it fell on its sword spectacularly. In fact the only kernels we really have, minus something like a PDA, are Tron, BSD, Linux and NT.

Now that software is at the level it is, the cost of just improving what we have is so large that to start from scratch is becoming more and more of a non-starter, for both OSS and closed-source software. In a lot of the closed-source stuff, this comes in the form of libraries Apple ships with routines developers can draw upon, because it's just stupid for every developer to have to implement spell-checking. However, more frequently it might well be an OSS library too, and Apple may well be incorporating an OSS library when it doesn't make sense for them to reinvent the wheel.

Even the best OSS stuff we have is often built on shoulders of giants. An example might be AdiumX (www.adiumx.com) or Fire (fire.sourceforge.net), where they wanted to create a kick-ass native chat client for the Mac. If there wasn't an OSS solution available for them to draw on—LibGaim—(encycl.opentopia.com/term/Gaim) we'd probably still be waiting for the first version to be released. Because it was there, they were able to write a smaller percentage of new code to do what they needed, and their efforts were multiplied by the available library and the fact that they didn't have to reinvent the wheel.

TSH: Aren't you forgetting Mach in your list of kernels?

DBM: I am of sorts, but Mach would be pretty lame without the BSD userland, so I just sorta lump them and BSD together in my head, as minus a few weirdities it feels like BSD to a user. Still out of all of them, if we're not talking about handhelds and such, NT is really the only major closed-source kernel around, and I doubt they'll be rewriting it anytime soon, especially since they were smart at the beginning and made the code very portable.

TSH: Who are the giants in the OSS world now? Which projects are the ones to keep an eye on, especially for OS X users? If you've been reading my column, you'll know that Patrick Luby is certainly at the top of my list.

DBM: What project is Patrick Luby on?

TSH: NeoOffice/J (www.neooffice.org).

DBM: Oh my gawd. Duh. I have an email from him in my inbox, that's how far behind I am. I feel so sorry for that guy. Many people have bitten off more than they can chew, but he and his partner really have the proverbial elephant. It's literally like two guys and 300 megs of code.

TSH: Why would you feel sorry for Patrick? He and Ed Peterlin are really the only ones who are positioned to do what they're doing. Patrick used to work at Sun on the OpenOffice Project and is a J2EEE wizard.

DBM: Simply because of the requirements they've put on themselves to do what they're doing—it's a Herculean task. Absolutely Herculean.

TSH: I think what they've done is spectacular. Absolutely spectacular. If Apple isn't paying close attention, it should. If they don't win something at WWDC, I'll be very very sad. All of the hacks that make NeoOffice/J work should be codified as libraries for XCode.

DBM: They have a project with a massive code base, doing a ton of interrelated things, that was most certainly not intended to be on a platform like OS X, so the amount of work they've set out for themselves—their final goal—would take a team of 20 a year to complete if not more. And they're two guys, with a project so large only they really understand it, so few people are willing, let alone able, to roll up their sleeves and pitch in.

The biggest 'projects' I keep an eye on are the force-multipliers, which have a way of sneaking up on you. Growl (growl.info) is a very good example of this, which I'm keeping an eye on for a few reasons.

TSH: I've done the best I can for NeoOffice/J from the sidelines. Rah rah rah. If Patrick and Ed could "bottle" what they did, it could bring a lot more applications to OS X.

DBM: If you've never heard of Growl, it's basically a unified event system for Mac OS X. You can set any up to throw its messages through it, instead of bouncing the dock or throwing up an alert. It's like the iChat bubble, except it fades away, so you don't have to switch to your mail client just to see what the new message in your inbox is.

TSH: Yes, I know what it is, but as far as OSS goes, it's for OS X only.

DBM: It's a really valuable service, so it won't stay OS X only for long. I'm told others are looking at implementing it almost verbatim on other platforms, as people are really digging it. They've gone from 19 apps supporting Growl when I first talked to them to over 120. People like Skype are dropping in support, now.

TSH: That's certainly going to help!

DBM: One of the things that fascinates me is what a force multiplier it is. Here's an app showing a great service that people love. Normally, those 100+ apps would all have to implement their own, instead they all get the functionality by tying into Growl. If it was closed source, they probably wouldn't have their app depend upon it unless it came from Apple, and Apple doesn't have anything like it.

TSH: OK, so a shared library=force multiplier in your view?

DBM: Well, all OSS code is a force multiplier in the sense that someone is saving time by using it instead of reinventing the wheel. In the case of OS X it's their kernel and userland, the CLI stuff, and in the case of Growl it's a way to throw messages on the screen. It's actually ingrained into the idea of OSS, and why we have licenses like BSD and LGPL instead of just the GPL. There are cases where it just kicks ass if more people are using something than if they aren't, and if the TCP/IP stack wasn't BSD, who knows what Microsoft or Apple would have used and how screwed up it could have made things.

Another thing I think is notable about Growl is how their developer system is setup, and I'm kind of fascinated to see how it'll work out as it contrasts to heavily with say, the NeoOffice/J situation.

\$32
< 1 Year



2 Years >
\$64

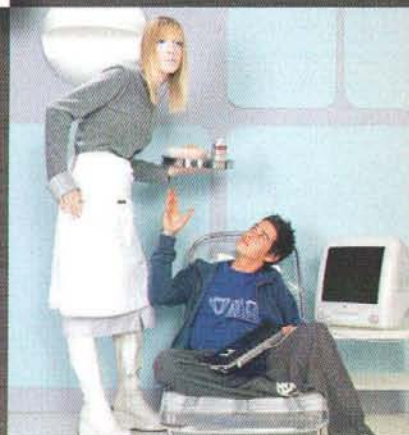


Interviews

Tapping into the world of celebrities and their Macs, only MacDirectory offers exclusive interviews. Get a close and personal view from Sarah Jessica Parker, Steve Jobs, Madonna, Harry Connick Jr., George Lucas, Jennifer Jason Leigh, Steve Woz and other leaders in the Mac community.

Features

Designers, writers, musicians, business leaders & our technical expert team offer their own personal interpretation of things that only the Mac system can deliver. With more than 200 pages of news, insights, trends and the largest Macintosh buyer's guide including over 5,000 Mac products and services.



MacDirectory

BEYOND ANY MACINTOSH MAGAZINE.



Culture

MacDirectory takes you to the wildest corners of the world and uncovers how Macintosh computers are being used by other cultures. Travel to Japan, Australia, Germany, Brazil & Russia and learn more about Apple's cultural impact around the globe.

Reviews

Find out all you need to know about the latest Mac products including the hottest Mac OS software and hardware.



Subscribe >

macdirectory.com

Send check or money order to:
MacDirectory Subscription Dept.
326 A Street, 2C
Boston, MA 02210

They have bounty of help, well over 20 developers from various other OSS projects, all pitching in on this one service doing things they want to see in it because their app will be making use of it. Normally in a situation like this, there's a tendency to go to lead developers, choke points who keep what goes into the software sane. They basically opted for a different approach, which is practically socialism in development with no real "no's" going around.

TSH: Growl is interesting to me in the sense that it was never intended to be an Application, but an extension to other applications. Sort of like the OSS version of an OS X Framework. I wonder what would happen, say, if Apple open-sourced one of their Frameworks, like CoreImage. I wonder if the code would lend itself to picking up by others, or would be considered a proprietary mess. It's an interesting hypothetical question.

DBM: Probably a little bit of both. I know a lot of the KHTML developers are not big Apple fans right now, or rather they're very annoyed with parts of the fan base who have the wrong idea about their relationship with Apple and Safari. One of the biggest problems with anything like that though are prior considerations.

TSH: Much of the same speculation's revolved around the source code for Hypercard, which people have been asking Apple to release. I say "be careful what you ask for, you just might get it."

DBM: Most of the licenses out there require you to give up the patent rights to whatever you release, so you can't start beating people over the head later, and there could well be things Apple doesn't want to give up. And truth be told, you never know what Apple actually owns. What I'm saying is, Apple just might not really have the option with some of their stuff. If we go back and look at NeXT, Apple was using Display Postscript for their display layer, which they'd licensed from Adobe. Assuming NeXT wanted to open source their OS completely, I am going to assume NeXT would have had a hard time getting Adobe to allow them to Open Source their Display Postscript stuff.

TSH: Well, yes, it's widely known (and even trumpeted by Tiger itself) that OS X (and QuickTime 7) contains royalty-bearing technologies.

DBM: Yes, that it does, many of them surrounding codecs. But most of Apple's software is a black box. We have no idea what's really in it, so we can't assume that there's nothing in Quartz they haven't licensed from other companies or don't own the patents to, so they can't give it away.

TSH: The "Black Box" analogy of Apple's software is more than a little interesting to me. In your recent deconstruction of Maui XStream's CherryOS, you used various techniques to compare memory dumps of CherryOS and PearPC (<http://www.pearpc.net>) to prove your case.

Of course, knowing in advance what to look for made your job that much easier.

Yet if we were to apply the same techniques to OS X, or Windows 2003 Server, and had the luck to know what we were looking for, do you think either would come up clean?

DBM: Feel like you're treading on dangerous waters? *grin*

See, here's the deal. Companies like Apple and Microsoft are

basically becoming intellectual property companies. It's their lifeblood now, it's just everything. People think of Microsoft as not being very innovative, but in some ways that's old-school thinking, and doesn't take into account just what they spend on research and just how many patents and such they're pumping out, and it goes the same for Apple.

TSH: Maybe. However, before I went into the computer business I wanted to be a poet. I wrote poetry, and studied for my Ph.D. in creative writing for six years. You read other people's poetry and you read. And you forget it, but it creeps into your writing. T.S. Eliot once said, "bad poets borrow, great poets steal."

Now you can get away with that in the arts. You can hear an echo of someone else's voice inside your writing and often that's desirable, yet there's no mechanism to make a byte for byte comparison of that. In the software business, there is. Especially when you know what to look for.

The reason I guess I'm bringing this up as Apple and Microsoft rigorously defend their IP, is the question of how well they'd fare if put to the metal?

DBM: Right, and there's that element, but not here. Their IP is their treasure-chest. Their current product might die, but their IP is what they live and die by, and to endanger that would be the height of stupidity—the potential cost is just not worth the risk. People say that the GPL has never been tested in an American court, but it's written so tightly that no one wants to try. And the larger the company, the larger the payout. It's hard to go after small to medium business, but lawyers start smelling dollar signs when you hit multinational and the words class-action are used.

TSH: Yes, the GPL's a work of "spare genius" (quoting myself) If there was any type GPL code in their product, and it was proved, who knows what would happen or what they'd have to do or how devastating it could be. They could lose all the patents that are in it, so there's just no point in taking that risk.

DBM: Now, you could well find code if you knew exactly where to look, but it probably wouldn't be what you think.

TSH: There's no way that Apple or Microsoft can control their code that closely. All it takes is one lazy programmer that wants to make it to their son's baseball game, and it's in there. No one is the wiser. OSS is so prevalent, that it has to get inside some programmer's head.

DBM: Exactly, and that's pretty much where I was going. If a programmer has a problem or deadline, and access to Google, they may well come across a function or algorithm somewhere and drop it in and no one is even aware. That stuff is going to happen, sure, and there might even be some commonality to it, but it's not the end of the world.

TSH: I have seen enough of Tiger to know that Apple isn't spending a whole lot of time scrutinizing every aspect of its own work. Tiger changed the output of diskutil and system_profiler, for example, which broke some of my scripts. Really the question in my mind becomes how many proprietary software products could stand to be scrutinized the way you scrutinized Cherry OS and pass the test. I wonder

how common it is to cobble together OSS code and sell it as your own.

DBM: I can't talk specifically about CherryOS and such, because of lawyerly-things, but while I think its a good question the answer always comes down to: quite a few, but rarely anything big.

TSH: If, for example, I were the CIO of a corporation, and someone came to me with a streaming video solution that worked quite well, and priced well, could I safely assume that it wasn't a fraud?

This seems to me to be the biggest question your CherryOS investigation has spawned, at least in my mind.

DBM: Right.

TSH: Shouldn't software be "checked out" before it's bought or sold? And who is going to do the checking?

DBM: If it were me, one of the things I'd look at was how well the software was known. This stuff doesn't really occur in the mainstream, simply because someone has too much to lose and it almost always end up coming out if someone mainstream tries it. As the exposure of an app goes up, and people become aware, they'll get suspicious if things aren't on the up and up.

TSH: I'm going to WWDC in a couple of weeks, and I'll be surrounded by many folks who I admire from the OSS world while simultaneously surrounded by those from Apple who code for the black box. In this information-driven world, wouldn't cross-pollination be the norm, rather than the exception?

DBM: OSS is starting to become so powerful because of economics, not ideals. The ideals are awesome, but long term it's just economics. Copyright law goes through the congress, they are the ones who would be legislating such a thing, and OSS doesn't exactly have a major lobbying arm.

TSH: Well you and I are forming a very weak arm, but maybe it'll get stronger.

DBM: I know, I've just had to learn so much about trade secrets and IP that its just a lost cause in this way, because to me the real problem is that OSS is actually very well-protected, it's just that so few of those can make use of those mechanisms.

TSH: I honestly believe that ripping off OSS is more common than you'd like to think.

DBM: Oh, I think its common and happens, just more on the fringes. As awareness of an app goes up, and more eyes are on it, suspicions are there and it's not hard to poke around—and the consequences can be devastating to a company.

And there are great mechanisms for protecting OSS, it's all built into the copyright law, but because it's a civil thing the resources that are often required to exercise them are just huge. Here's an example, the vast majority of OSS developers never go out and register the copyright for their code. This costs all of \$30.

TSH: Well, they should!

DBM: So let's say they find out a company is basically taking their code and reselling it, and want to go after them. A good IP lawyer costs \$150 to \$400 per hour or *more*, and most individuals simply can't afford that kind of cash. However, if you've registered your copyright before someone is distributing stuff with your code in it, it triggers all kinds of things like "willful infringement" and damages are starting at \$100-150k and

up. Now it starts making sense for an IP lawyer to take your case on contingency for getting his cash from the damages. . . but no OSS app developers do this, because this is fun for them, not a business.

TSH: So it costs an OSS developer just \$30 to copyright their code, but how much does it cost Apple to patent something? A whole lot more! So I understand exactly what you mean. But the OSS programmer has another recourse, one much more influential than an IP attorney. . .they contact Drunkenbatman and cry tears of grief into the White Russian on your home page. . .



Figure 2. The Drunkenblog White Russian

DBM: Lol There is that option, but it's not one I'm encouraging. Lol sorry, I'm cracking up here.

TSH: But let's take this example. Not only is there the legalities of "willfull infringement. . ." I would, as the developer, take my case directly to the customers of the folks hawking the product with my code in it . . . I think they might be a little concerned, at least as far as their image goes.

DBM: Yeah, but while I'm not a lawyer, I believe there can be legal problems in doing that too. And really, the USA and such isn't where the scary stuff is going on. You want to be scared, drop into China or Pakistan or Russia. This isn't well known to Mac users, because no one there uses Macs, but it's just a nightmare there in terms of IP.

Intellectual Property laws have gotten a bit whacky here in the USA and such, but when you drop into one of these countries with almost no IP laws to speak of, all bets are off.

TSH: Microsoft especially has spent a fortune on trying to clean up Windows Piracy in China, and is succeeding, I hear.

DBM: Barely, and China is just one of them. Kyrgyzstan is a place I've learned a bit about, and you basically pop down to their markets and anything you want costs \$5 on a disc.

TSH: The whole notion of IP is pretty much a Western ideal anyway.

I've done quite a bit of traveling, and trying to find "genuine" software, music, or DVDs in certain countries can be quite difficult. Getting the rest of the world to subscribe to our views of IP is going to take generations, if we're successful at all.

DBM: There's simply no recourse in these countries, and while IP laws are whacky here, they're what allow open source in the

first place. You drop in on some of these Windows shareware sites, and there are so many red flags around some of these little utilities. But again, Mac users don't really see this because hell, Macs are so fricking rare outside of a major urban area, let alone California. Half of their sales are to the USA, so half of their sales are spread out across the rest of the globe, which means they're spread pretty thin.

Oh, other countries will get there as they become more prosperous, they simply have to. It's economics. Corporations are starting to funnel a lot of their IP into countries like China, and they want that protected. And as they generate more of their own IP, they'll want it protected. They just aren't there yet, but they will be, and it'll happen faster than it happened here.

TSH: Here's a question: would open-source even exist if there were no IP laws?

DBM: Yes, but only in the form of "public domain software".

TSH: Well I think you and I agree that poaching OSS software should be met with consequences.

Whatever they might be.

DBM: OSS is basically someone saying "I'm going to put this on the shelf for you to use, instead of keeping it to myself, but only if you follow the restrictions in this license." If you remove the ability to put those restrictions in, you're just left with "I'm putting this up here, do whatever the hell you want."

And we've always had things like that in some form, but it would be much much less than the sheer mountains of code that are being funneled into OSS.

TSH: Essentially the only restrictions are (generally) 1) distribute the source with the product 2) all rights you have are everyone else's rights.

DBM: Right, the GPL is designed to keep what's free free.

I do think OSS should be protected, but it's basically about copyright law and our system. And there's no way in hell I think I'm smart enough to fix our system, or even offer suggestions. . . I just know it's very hard for an individual to protect his intellectual property, and that it's primarily the domain of corporations.

TSH: Yes, free in the sense of uncaged. . .

Not free as in the sense of cost

DBM: Yeah, that's free as in beer versus free as in speech. There's no problem with selling OSS code as long as you don't take away rights or impose additional restrictions. It's a remarkable document, and was put together by very smart people, although reading it can make parts of your brain ache.

TSH: I know, my February column was largely about the GPL.

DBM: The spirit of the GPL does match the legalese, but the spirit is about keeping free software free. However, a lot of people have never really read the GPL, but have the spirit of the GPL in their heads as something a little bit different.

DBM: This is where the problem with Apple and KHTML comes in — they're following the legalese just fine really, but many don't think they're following the spirit.

TSH: Oh Apple and KHTML. Yeah, that's a good point. Do you watch *The Bachelor*?

DBM: LOL, no, I can't say I watch *The Bachelor*. There's a ton of good TV out there, but I generally go by what's

recommended to me, and *The Bachelor* has yet to be recommended. Should I be watching?

TSH: Well, my girlfriend and I watched the last season of *The Bachelor*, and it seems that the guy picked younger, less mature women because they're less of a threat and will be more compliant. I see the relationship between Apple and KHTML as somewhat similar. I think Apple feels that the KHTML folks should be saying "thank you thank you Apple for selecting us, without you we'd be nothing . . ."

Many people questioned Apple's choice of KHTML over Gecko, I was certainly one of them. But if you look at the Mozilla Foundation (<http://www.mozilla.org>), that's something, not just a little open-source project. I think Apple felt it could have its way with KHTML, but now there's trouble in paradise. . .

In many respects, despite how far Safari has come, I can't help but feel that Apple's wasting its time because it seems that Firefox is rising, while Safari is simply bundled.

DBM: Really? Well, there are certainly wham-bam-thank-you-ma'am aspects to Apple, but most of that is related to your pocket book. In the case of KHTML, and why they chose it, you really have to go back to what their options were at the time.

TSH: They had another, more viable option. But they really needed a browser. Badly. And quick.

DBM: Here's the thing: Apple is really short on people. They give the impression of this large company, but they are really starved for resources—their Mac division could well be in the red for all we know.

TSH: I understand that partnering with an OSS group is a form of thrift, but it would've taken less to slap Camino into shape. I think Apple took the path of least resistance.

DBM: The Mozilla engines are massive. They're multi-faceted, and weren't just a browser engine, they were a platform engine. The entire interface is rendered in them, with technologies like XUL. The code base was just this huge, massive, spidered mess of code, and everyone on those projects knows it. It's similar to the NeoOffice/J situation—getting your head into it is hard. Wrapping your head around it with limited resources would be a nightmare, and my hat is off to Mike Pinkerton of Chimera and Camino fame in more ways than you know.

There aren't that many people working on Safari. If I recall, OmniGroup tried to take the spidermonkey code for the Javascript engine in their OmniWeb browser, and it was just a nightmare. KHTML has major deficiencies in terms of compatibility and tech, but from a time to market and resources issue, I wouldn't be surprised if KHTML was really their only viable option.

I think Pinkerton would say the exact same thing, that it's completely understandable why Apple went with KHTML over Gecko and friends, especially if they knew they were going to be embedding it into the OS and doing things like Dashboard.

TSH: For some reason, I even knew back then that something Mozilla-based would eventually become the dominant browser on OS X. As far as timing went, yes, I agree that KHTML might have been the best option at the time, and OS X users were truly desperate to see something other than IE.

I think the KHTML choice was an expedient choice made at the right time. In terms of long-term viability, I don't think it's going to work out.

DBM: Has Gecko become the dominant engine on OS X?

TSH: As far as I'm concerned, it has.

DBM: I wrote about this a bit in "Gecko's Unsightly Ass" awhile back. KHTML may have been their real viable choice, but yes I'd agree that Firefox is a better browser than Safari—although now necessarily on OS X.

TSH: Not for home users, but certainly for the corporate world. Firefox has been a Godsend for IT managers because there's a PC version.

DBM: This is actually a really interesting thing on the Mac. Per capita, we have just tons of OSS around, but something like Firefox is just a bit deficient on the Mac because they don't have the Mac people working on it. Thunderbird kicks ass on the PC and Linux, but is just kinda weird on the Mac.

Windows has a ton of OSS code around, way more than the Mac. However, per capita, while it's growing, I don't think it's as large as it is on OS X and Linux. If Apple hadn't shipped a free toolchain, I'd wonder if it would have the OSS community in the way it does now.

TSH: IT wants Firefox because it's not as insecure as IE, and because java stuff will work reasonably well cross-platform. Safari barely works with jsp or applets though it's getting better.

DBM: The key for Apple then is having to 'keep up' with the standards, and devoting resources to it. Firefox has nailed out a lot of these compatibility issues, so if they were using Gecko they could "ride their work", but again I think many think of Apple as having more resources than they do. Safari unfortunately doesn't work with a lot of things. It's important not to dismiss it, because Hyatt and others have done some massive and impressive work there. However yes, the onus now is all on them to improve Safari and bring it up to par.

TSH: I'd like your take on the VideoLAN Project, whose developers I affectionately call the "Coneheads."

DBM: The VideoLAN project is an awesome one, and ties into some of what I was talking about earlier in a major way. They're really screwed on OS X.

TSH: What do you mean, "screwed?"

DBM: They simply can't get the people with the expertise they need to work on the project. Remember, it's a cross-platform app, but there are almost always platform-specific things that need to take place in order to have the app kick ass. My understanding is that right now, they're having major, fundamental problems dealing with CoreAudio and other technologies. Which means they need people with real CoreAudio experience to help, and those are about as rare as little boys at Neverland right now.

TSH: Oh, you mean they're screwed as far as Tiger goes.

Not just Tiger, although that's a big stumbling block. There are issues with the current app on 10.3 they know about. Things that aren't in that they want to have in it, things that are in it that are flakier than they should be. However, VB developers are a dime a dozen on windows. Developers aren't a dime a dozen

on the Mac, let alone people who really know what they're doing, let alone people who know what they're doing and know Apple's technologies, let alone people who are all of the above and have the time to contribute.

TSH: Well, I've been pretty impressed with VideoLAN on 10.3.

DBM: I didn't mean to imply I wasn't, just talking about the situation they're in. It's an extremely impressive effort.

TSH: I wouldn't be surprised if someone didn't step up to the plate to help them out. VideoLAN client saved OS X for many people, including myself.

DBM: It saved it for you?

TSH: In many ways, it's VLC that makes OS X tenable as a consumer OS.

DBM: There's always MPlayer. . .

TSH: Oh it's OK, but VLC's a thing of beauty

DBM: *grins* Just saying, you might be surprised at how few people are actually working on the OS X port of VLC, which is why it's so impressive.

And yes, QuickTime isn't exactly the draw it once was to consumers. And is especially onerous in 10.4, but perhaps that's just me.

TSH: I think I heard from all two of them when I wrote my first article "How the ConeHeads Could Save Apple" (<http://www.themachelpdesk.com>) in 2002. And hey, you get 100% of your daily "Royalty-bearing" technologies when you use QuickTime Player Pro. . .

DBM: I lurk on some of what they're doing so I can have a finger on their pulse, and while I can't say the exact number, I'm pretty sure I can count the developers on one hand while holding a mug. And yes, you do get your daily allowance. And they make sure to remind you of that, everywhere, while asking for your cash so you can play your movie full screen. It's really egregious and uncool behavior, and this is one of the things that can just set me off. They're theoretically in the middle of a fricking media player war with WMV and Real, yet they're making their player just suck. We saw this with Netscape and the browser wars too. Microsoft certainly did uncool things, but I'll go to my grave saying one of the reasons why Internet Explorer won, if not the main one, was that Netscape just started to suck.

TSH: Let me see, I use VLC for: 1) Streaming instructional videos for customers 2) playing back content archived from my ReplayTV 3) watching VCDs 4) Watching .asf files 5) ripping incompatible video formats to raw pixels to import into QuickTime. VLC's the Dutch Boy in the dike of OS X, as far as I'm concerned. Plugs lots of little holes.

DBM: I guess I'm a little bit of a newbie when it comes to VLC, I pretty much just use it for watching Doctor Who.

TSH: And I know from reading your blog that that's really important to you. . .

DBM: If VLC didn't exist, QuickTime does accept plugins so those codecs could be dropped in like 3iVX and DiVX are doing, but VLC just does it so much nicer.

Good entertainment is a thing of beauty, and should be acclaimed. Good entertainment with an element of random

absurdity that only the British seem to be able to really do just makes me smile. Lol, QuickTime is, at least from what I've told, something you do not want to see open-sourced. The code is just monstrous. Ok, I'm starting to feel bad. It should be mentioned that I have great respect for Apple and much of their work, but just that they've been pushing the edge and letting lots of things pile up that are starting to come to a head.

Maybe we could talk about something positive. Like iPods or something.

TSH: Well, Tim Monroe, who's a columnist here at MacTech, shows bits and pieces of QuickTime's capabilities in each month's column. QuickTime is certainly a whole lot more than a simple video player—plug—for example, if you wanted to develop a video interface to a hardware device, QuickTime has facilities for that.

DBM: QuickTime is a platform, that's been built up over years and years, and as such is kind of like Mozilla—unless you've been working with it for years and years, your brain would just go into a coma if you had to deal with the code.

TSH: You mentioned that the VideoLAN Folks needed programming help. They might need legal help as well. It seems that the European Union is trying to shut them down. How do you feel about such a valuable (especially to OS X) OSS technology being subject to targeted legislation from a governmental body to uphold patents, even when it's obvious that no infringement has taken place? Is a format the be-all-end-all, or can a format be successfully arrived at from two separate code bases?

DBM: As to the legal issues around VLC, I'm assuming you're talking about the proposed and pending changes to the European IP laws to make them more in line with America's?

TSH: That's precisely what I am talking about.

DBM: It's a rough situation, and one I haven't really made up my mind on. See, there are different kinds of infringement at work here. There's copyright, which is an expression of something, and then there's a patent, which is an *idea* of something.

TSH: The VideoLAN folks seem to be quite convinced this might be the end of the road for their project.

DBM: It could well be. The problem isn't the code they've written, that's theirs. It's the ideas covered by the patents in many of the things they've written: namely the codecs.

I should say that people saw this coming, and it's why we have formats like Ogg, which were designed from the ground up to not infringe on any patents whatsoever.

TSH: Well, when something is distributed in a certain format, does the patent follow the product indefinitely? So what you're saying is that the actual video product can be irrevocably closed, so that it can never be legally viewed without a licensed codec?

DBM: Ogg is the container format, with various codecs, audio and visual, and something like Vorbis would be the audio codec. It's completely patent-free, but Ogg Vorbis hasn't taken the world by storm for audio files.

It's not about the format necessarily, it's the ideas that go into creating the format.

Say for example, something like H.264. This codec was basically

designed by a bunch of companies that got together and pooled their IP to create this killer codec. The key word is IP, and they own the patents to all of it, and setup a company designed to collect the royalties. Same for MP3. Lots of Linux distributions require you to download an MP3 player, or perhaps it's the encoder, because they can't do it legally. Apple went and bought licenses to all these things, which others aren't.

TSH: Well, that's a gray area to me. For example, Microsoft got away with FPNW (File and Print Services for Netware) and a Gateway Service for Netware, which helped Windows NT displace Netware as the dominant Network OS in the corporate world.

That was OK, right? Samba is OK?

TSH: SAMBA and such are different, that's clean-room reverse engineering.

And how is VideoLAN not reverse-engineering?

DBM: Because they haven't done any reverse-engineering. With something like SAMBA, they setup a Windows machine, sent it packets on the port, and tried to guess what it expected. After a ton of trial and error, they were able to work out—with some bugs—what NT was expecting and such. To my knowledge, patents weren't an issue.

TSH: And so being able to play back a format without the source code to the codec is done how?

DBM: What goes into creating that format is a published spec, what Apple's calls an "Open Standard", but it's really a misnomer. This is usually in relation to Microsoft's codecs. It's not an open spec, it's one they've put up to use and use, *provided you pay them what they're asking*.

And really what it comes down to are things like method patents, and through them software patents.

TSH: What you're implying is that the VLC folks are knowingly NOT paying what they're supposed to?

DBM: Well, if they were in America they'd probably be shut down already, and that's the problem. Yes, if they are going to be using someone else's IP, under our system, they should be paying.

TSH: So what you're saying is that if the EU changes IP law to be more like ours, the VideoLAN project will have to move to China or pay.

DBM: Lol. Pretty much. I doubt that the code will die, this stuff is like playing whack-a-mole with a hydra. It'll go splat, spread, and reemerge all over the damn place.

Right now, the EU is lacking some things we have here in the USA around software patents, and that's really where the problem is.

TSH: Well, you've kind of shot down one of my favorite projects. Damn you Batman!

DBM: Once those IP laws go into place, there'll be a bullseye on VideoLAN in a big way.

No, like I said. You can rebuild it, and make it stronger. It may even make weird noises when it runs, it'll just be doing it in Norway or China or something.

Now see, lots of people think software patents are just wrong, and they may well have a good point.

TSH: Yes, I would be one of those, especially in those cases when proprietary software won't make nice with others,

but that's a can of worms we won't open right now.

DBM: We have some really, really stupid shit going on. A good example is Amazon's "One-click" patent, which well, gave them a patent on being able to click a button and buy what you want. Apple ended up licensing it for \$1 million dollars. Or someone ends up with a patent on a type of Window, or having something in a specific place. It doesn't matter how it was coded, or the implementation, it's just the idea itself, and people much, much smarter than myself think we should really be looking at how these laws translate to our software.

TSH: I'd really like to know where Drunkenblog is going to take us OSS fans in the next year or two.

DBM: Yeah, well I've set the bar kind of high now haven't I? Deconstructing MXS took well over 150 hours.

TSH: Yes, you have my friend.

DBM: my plan is back to business as usual.

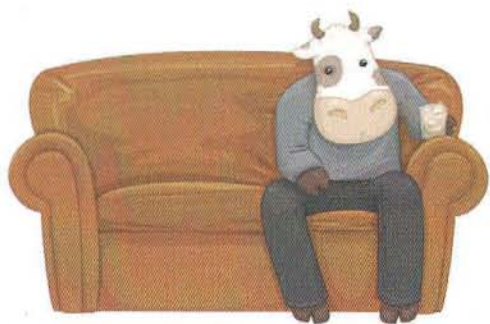


Figure 3. Business as Usual

In Next Month's "Source Hound"

Well, after my long talk with Drunkenbatman, I think I need to retreat a little to something lighter, so I'm going to check out an Open-Source Project that everyone, including Apple, absolutely loves: SquirrelMail. That's right, the ubiquitous, fun and nutty piece of webmail magic.



About The Author



Dean Shavit is an ACSA (Apple Certified System Administrator) who loves to use a Mac, but hates paying for software. Since he's not into breaking the law, his most common response to any cool solution is: "Does that cost money?" If it does, you can bet he's on the hunt for an Open-Source or freeware alternative. Besides surfing for

hours, following the scent of great source code, he's a partner at MOST Training & Consulting in Chicago, where he trains system administrators in OS X and OS X Server, conducts large-scale Mac Deployment and Upgrade projects, and writes for his own website, <http://www.themachelpdesk.com>. If you have questions or comments contact him here: dean@macworkshops.com.

Effortlessly edit your PDFs



PDFpen

Now you can fill out and save forms,
split, combine, search and even
scribble on your PDFs with ease!

"4 mice" – Macworld

\$49.95 - download free demo at PDFpen.com

Also available: **PDFpenPro** for creating fillable forms



Smile
on my mac

smileonmymac.com

Creative software for your Mac
that does what you want!

INDEXING DARWIN FILES

EXTENDING SPOTLIGHT'S REACH

BY RICH MORIN

Spotlight is quite willing to search the user's home directory tree. It also looks into some other areas, including /Applications, /Developer, /Library, and /System. It stops short, however, at looking into the "Darwin" directories (e.g., /bin, /private, /sbin, and /usr).

As a result, many interesting files (e.g., control and include files, man pages) are not indexed. This seems like an unfortunate limitation. Most users aren't interested in these files, but some of us (e.g., administrators, power users, and programmers) find them to be very useful, indeed. Wouldn't it be nice to be able to use Spotlight to index these files?

In fact, it is possible to override Spotlight's built-in restrictions. Each volume's root directory contains a sub-directory (named `.Spotlight-V100` in Mac OS X version 10.4). This contains a few "database" files and a property list file (`_rules.plist`). By editing this property list, you can instruct Spotlight to include or exclude directories.

Just locating a file isn't enough, however; we also want to be able to view it. And, if the computer can find connections between the files, some navigation aids (e.g., hyperlinks, linkage diagrams) would be lovely to have. Although the interface could be implemented in Cocoa, HTML is an arguably simpler (and far more portable) approach.

As an experiment in extending Spotlight's reach, I decided to write Morinfo, a suite of scripts that allows man pages and include files to be located by Spotlight and presented by a web browser. The results are useful, even in their current state, and offer interesting possibilities for future enhancements.

Background and Competition

Turning man pages into HTML is not a new idea. There are existing facilities (e.g., Internet servers and downloadable applications) that perform this function. However, their approaches (and consequently, results) vary from mine.

The FreeBSD Project (<http://www.freebsd.org>) provides access to the "FreeBSD Hypertext Man Pages" (<http://www.FreeBSD.org/cgi/man.cgi>). Despite their name, these pages cover a variety of operating systems (including Darwin). Of course, the site cannot display arbitrary man pages that a user might have installed locally.

The pages are created by a free (i.e., libre) Perl CGI script, available via the FAQ. The script runs the `man(1)` command on an appropriate input file, reformatting the results into HTML. Search is available within the CGI; Internet search engines (e.g., Google) also index the pages (e.g., FreeBSD `chmod`).

Bwana (<http://www.bruji.com/bwana>) is a free (i.e., **gratis**) application which bills itself as a "man page viewer for Safari, Firefox, Camino & IE". To use Bwana, the user enters a URL into one of these browsers, using the scheme `man` (e.g., `man:chmod`). Bwana puts the generated HTML in a temporary file (e.g., `file:///...`) and tells the browser to display it.

Because Bwana generates its web pages as files, it cannot support a forms-based "search" facility. And, because the files are temporary, Spotlight will not index them. Bwana does, however, provide an "index page" (accessible via `man`;) which lists the known man pages (with short descriptions, if available).

Both of these facilities provide links to other man pages, but only within a limited context. Bwana only links to pages on the local machine; the FreeBSD site only links to pages for the current OS variant and version. Neither facility links to anything other than man pages (e.g., include files).

Objective and Approach

My initial objective was to make the files convenient to use and searchable by Spotlight. The usage requirement was met by handing the files (suitably edited) to the user's default web browser. The "search" requirement was met by (a) using an extension that Spotlight indexes and (b) putting the files into a directory that is traversed by Spotlight.

HTML makes it possible to generate hyperlinks, using implicit linkage information that is present in the files. The entries in a man page's "SEE ALSO" section, for example, can become links to other man pages. References to include files (e.g., in sections 2 and 3) can also be turned into links.

Even with the addition of client-side technologies (e.g., Java, Javascript, SVG), web pages do not compete with Aqua for attractiveness, interactive performance and richness, etc. On the other hand, they have significant advantages for this (exploratory) project.

HTML gives me a familiar and powerful user interface, with both local and Internet-wide navigation. Better yet, it requires very little effort on my part. Finally, HTML is a much more portable technology than, say, Cocoa.

Resource Usage

Careful Reader may have noted that I haven't discussed the resource usage of this approach, whether in online storage or processing time. This is not because there isn't any; indeed, it uses hundreds of megabytes of storage and (initially) several hours of processing time. However, I don't think this will matter for most of our readers.

The storage impact mirrors, almost exactly, the storage used by the original files. I add a tiny bit of HTML, of course, but this mostly gets lost in the noise. The edited man pages and include files (including BSD, Cocoa, X11, etc.) occupy about 350 MB of storage. Even if the edited versions used a gigabyte, most users wouldn't notice the impact.

The processing time, as noted above, is quite substantial. On a B&W G3, the initialization script runs for an hour or so. Spotlight's importers also take a while to index the material. On the other hand, these activities have little impact on the interactive performance of the machine, so what's the problem?

Online storage and background processing time, in the amounts Morinfo uses, are essentially free. If having a nicely indexed and cross-linked set of reference pages is useful, most users will gladly provide the necessary resources.

File-based HTML

My first attempt created HTML files in a folder that Spotlight is known to search. Mac OS X "knows" that files with an extension of `html` should be viewed by a web browser, so double-clicking these files "Just Worked". Unfortunately, this approach has significant limitations.

When an HTML file is double-clicked in the Finder, the resulting browser window shows a URL of the form `file:///...`. This indicates that the browser is reading the file directly (rather than through a web server).

Because no web server is part of the process, HTML files cannot make use of CGI scripts, HTTP header magic (e.g., cookies, redirects), Server-Side Includes (SSIs), or other server-based technologies. They can, however, take advantage of many client-side features (e.g., CSS, images and client-side image maps, Javascript).

An HTML file can also link to any type of web page, located anywhere on the Web. So, for example, it could link to a CGI-based page that post-processes the file itself, adding information or dynamic capabilities. I considered this approach, but (fortunately) found a cleaner solution.

CGI-mediated HTML

I now create "template files", using a CGI script to post-process them into web pages. This is more complex than my initial approach, but it allows me to use the full power of the web server. The following diagram summarizes our data and control flow, in the context of Spotlight and the Finder:

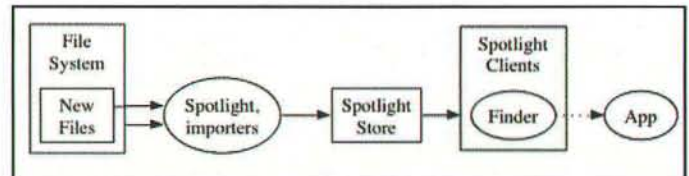


Figure 1. Data and control flow

Spotlight and its importers gather information from the file system and from new (or changed) files. This information goes into the Spotlight Store, where it can be accessed by Spotlight clients (e.g., the Finder). Similarly, `m_init` reads include files and manual pages, generating template files for use by `morinfo`.

The template files are "new files", stored in an area that Spotlight indexes, so any keywords found in them are entered into the Spotlight Store. This means that the user can find relevant template files, using the Finder's Spotlight mode.

If the user double-clicks on a template file, `m_link` gets called into action. Using the "POSIX path" of the template file, it directs the user's browser to display a generated URL. This causes `morinfo` to start up, read the template file, and produce a web page.

m_link

`m_link` is a tiny snippet of AppleScript, but it performs a vital service. When a template file is double-clicked (i.e., "opened"), `m_link` generates a URL that (a) invokes the CGI script and (b) includes the path to the template file.

The initial version of the AppleScript code below was provided by Chris Nandor, of Mac::Glue and MacPerl fame. The `on open` block is run when the script is asked to open a file. The enclosed code retrieves the file's "POSIX path", appends it to a prefix string, and hands it to an open location command:

```
-- m_link
on open this_item
    set filepath to POSIX path of this_item
    set pre to "http://localhost/cgi-bin/"
    set pre to pre & "morinfo?file="
    open location pre & filepath
end open
```

Launch Services

When a file is double-clicked, Launch Services attempts to determine which application should be "launched" to work on it. Traditionally, Mac OS files had Creator and Type codes (stored in their resource forks) which supplied this information. More recently, Apple has allowed file extensions to be linked to particular applications.

Because my files are managed by command-line tools (e.g., `perl`, `vi`), creating and maintaining resource forks would be quite a nuisance. So, I use the second method, asserting priority over any file with a custom extension (`.ht4m` – "HTML Template for Morinfo"). Any user can assert this priority, using the Finder's File > Get Info dialog, but the process is a bit involved:

- Click on a representative file, selecting it.
- Type Command-I (or use the Finder's File > Get Info menu item).
- In the Open with: section of the dialog, navigate to Other..., then to the desired application.
- Click on the Change All... button, telling the system to use the same script for all files with this extension.
- Close the dialog.

I didn't want to burden our users with all this effort, just to get `Morinfo` installed. So, I put some assertions into `m_link's` `Info.plist` file:

```
<key>CFBundleTypeExtensions</key>
<array>
    <string>ht4m</string>
</array>
<key>LSIsAppleDefaultForType</key>
<true/>
```

My installation script copies a "Morinfo" directory into `/Applications`, using `CPMac(1)`. During the course of this copy, the OS examines `m_link's` `Info.plist` file and notes these assertions.

Template Files

My template files are (typically) stored in `/Library/Documentation/Morinfo`. To support the widest possible range of initial files, I append the full path name of the initial file onto the base path. I also add an intervening level (e.g., `HF`, `MP`), to designate the "name space" (e.g., Header File, Manual Page) that is being displayed.

We also adjust the file names in assorted ways (e.g., removing any `gz` extensions, adding a consistent extension of our own). The resulting path names look like

```
/Library/Documentation/Morinfo/HF/
usr/include/stdio.h.ht4m
/Library/Documentation/Morinfo/MP/
usr/share/man/man1/cp.1.ht4m
```

The editing of the file content depends on the format of the initial file. In the case of include files, I simply create a navigational header and enclose the include file text in a `PRE` block in an HTML page. I also insert some "placeholder" strings for the CGI script to expand, as appropriate. Finally, as discussed below, I generate off-page links.

In the case of man pages, I also need to (expand and) format the pages into an ASCII representation. Fortunately, the `man(1)` command does this quite readily. I then turn any `nroff(1)` backspace sequences (e.g., `f^Hfo^Hoo^Ho`) into appropriate HTML sequences (e.g., `<B>foo</B>`).

Link Generation

The final step is to recognize references to other pages and convert them into links. Include files and man pages have fairly predictable formats, so recognizing `SEE ALSO` entries and `#include` references isn't particularly challenging. Conversion into links, on the other hand, can be.

Because of Darwin's eclectic nature, man pages have somewhat irregular names (e.g., `ls.1`, `cal.1.gz`, `tic.1m`, `md5.1ssl`). When I adjust the names of the man pages, I normalize these names. This simplifies the process of generating links to man pages, because I don't have to keep track of oddball extensions.

Generating links for `#include` references is where things get sticky. References don't specify the exact directory where the include file is located. At most, they provide hints (e.g., specifying a parent directory or a hint about the file's Framework). Also, many include files enclose `#include` references within conditional blocks.

None of this presents a problem for the compiler; Xcode and/or `make(1)` files keep track of include directories and pre-processor definitions, handing them to compilers as needed. In my case, however, I have no such help.

`m_init` deals with the problem in a somewhat "heuristic" manner. First, it scans directories where include files reside, recording the full path names of all files found. Armed with this information, it can make "educated guesses" about the path name that matches a given "hint".

If a conflict arises, an error message is generated. This allows the programmer to add an "override" rule to deal with the ambiguity. No, it's not very elegant, but ambiguous information is a common problem in automated metadata processing. Elegance is nice, but functionality is the overriding concern.

Incidentally, the use of offline processing allows me to do some cute things with the link information that I detect. For example, I could add "back links" (e.g., from an include file's page to any pages that reference it). I could also generate "context diagrams" to assist the user in understanding the (local) link structure. These features are currently under development.

Contextual Issues

By double-clicking on a file, the user jumps into a different "context". In our case, the new context is a web page (e.g., in Safari). This can cause confusion for the unwary: if the user makes a follow-on search, using the browser's "magnifying glass" text area, she might be expecting a Spotlight search to occur.

Imagine her dismay when a Google (i.e., Internet-wide) search happens, instead. To be fair, the browser's magnifying glass is accompanied by the word "Google". On the other hand, the user might be busy, or distracted, or...

Even disregarding the issue of confusion, how is the user supposed to search local files from within our web page? The "obvious" answer is to provide a forms-based interface to Spotlight, but this might cause even more confusion. Sigh.

Privacy and Security

The first incarnation of Morinfo copied (edited forms of) system files to a different location on the local file system. This did not introduce any obvious security holes, but it wasn't very powerful, either. The current version is more powerful, but it introduces some possible privacy and security holes:

- Web server – Web servers can have exploitable bugs. Administrative errors and oversights can allow security holes.
- CGI script – CGI scripts can have exploitable bugs.
- Publication – Publishing local files on the web may reveal private information.

If the server does not need to be visible from other machines, a firewall can provide some protection. Mac OS X has

a built-in software firewall which should be turned on and configured to provide the desired balance between security and convenience. Unfortunately, at this writing, it does not allow Personal Web Sharing (i.e., Apache) to be blocked. Strong protection can be provided, however, by the use of a 10-BaseT "router" (e.g., using NAT).

Some applications (e.g., CUPS) provide their own HTTP servers, using different ports than the default 80. So, a firewall can block access to CUPS, while allowing access to the "normal" web server offerings. Unfortunately, there is nothing to prevent two programs from claiming the same port number.

CGI scripts receive untrusted information from queries, cookies, etc. Scripts should check all incoming information, making sure that only legitimate requests are honored. The "Taint checking" feature (e.g., in Perl) can ensure that the script has done this, but it cannot tell whether the script's testing code was bulletproof.

Although read-only access to "Darwin files" seems pretty innocent, complexities can arise. For example, a developer might put proprietary files into a "system" area, then be unpleasantly surprised when the files show up on Google.

Current Status

The current version of Morinfo is largely a "proof of concept" implementation of a semi-mechanized tool for documentation collection and mechanization. The functions it employs or provides include:

- Collection (e.g., File System to Spotlight Store) – Spotlight (and its importers) perform dynamic collection of metadata about files and the file system. Morinfo analyzes and reformats two types of files: man pages and include files.
- Search (Query to File List) – Spotlight (and its clients) use the Spotlight Store's metadata for analysis, search, etc.
- Launch (File to Application) – Launch Services provides assorted ways to launch an application, based on the attributes of a double-clicked file. Morinfo uses this to bind its generated files to an AppleScript (`m_link`).

Morinfo uses `m_link.app`'s `Info.plist` file to assert jurisdiction of all files which have an unusual extension: `ht4m`. Some Mac programmers dislike this technique, but it has the advantage of not requiring the creation and maintenance of resource forks.

- UI Startup – `m_link` adds a preface string to the "POSIX path" of the double-clicked file, creating a URL string. It uses this to "invoke" an arbitrary application (the user's default web browser), directed to a particular document (i.e., the file).
- Presentation – `morinfo` (the CGI script) uses the file as a template, adding eye candy, navigational aids, etc.

- Feedback – morinfo's web pages include mailto links whose Subject lines reflect the current topic, etc.

Connections

To review a bit, Morinfo makes use of some interesting "connections". Mac OS X provides some of these by default; others require a bit of hacking:

- File content is connected (via importers) to the Spotlight Store.
- The Spotlight Store is connected (via the Spotlight APIs and clients) to user queries.
- File icons (e.g., in query results) are connected to "appropriate applications", as specified in Get Info dialogs, Info.plist files, etc..
- An AppleScript application can launch another application, feeding arguments to it as needed.
- A CGI script can connect system files to manual pages.

Because of "network effects" (http://en.wikipedia.org/wiki/Network_effect), each new connection takes advantage of existing connections (and makes them more powerful). At the same time, new connections may expose existing resources to new security holes.

Wish List

There are any number of directions in which this work could be extended. Let's begin with the functions listed above:

- Collection – Morinfo needs to have an importer, in order to track changes and additions to its monitored file areas.

Spotlight tracks files, but not processes. If Mac OS X supported DTrace (Sun's Open Source data collection system), it could collect metadata at arbitrary system call boundaries.

- Search – Spotlight's query language is missing features such as regular expressions and "order" comparisons for strings. The Finder's "Spotlight mode" lacks convenient ways to specify compound matches or exclusions. As programmers and users make their wishes known, we can expect to see some enhancements.
- Launch – The current (extension-based) link has philosophical problems for some. There may be a "cleaner" way to do this.
- UI Startup – m_link is acceptably sprightly (even on a G3!), easy to modify, and quite flexible.
- Presentation – morinfo is almost a "blank slate" at this point. I foresee diagrams (for context and navigation), interaction support, and much more.
- Feedback – Morinfo needs a wiki-like subsystem to collect page-specific user feedback. Sharing of locally-collected data (e.g., for trend analysis) is also possible.

MI

About The Author

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1986 (when he helped Apple create A/UX 1.0). When he isn't writing this for MacTech, Rich provides documentation and programming services, specializing in mechanized document production. Feel free to write to Rich at rdm@cfd.com.

Connect • Communicate • Collaborate • Securely

"Let me just check my calendar..."

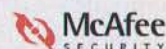


Kerio MailServer

A groupware alternative to Exchange that syncs calendars, contacts and email with Entourage and Outlook. Integrated anti-spam and McAfee virus filtering provide secure, junk-free email for users on any platform.

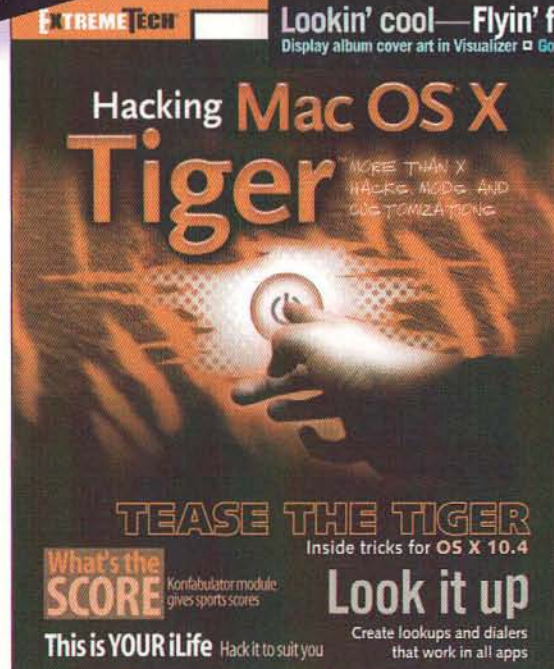
Mac OS X • Linux • Windows

www.kerio.com



Extreme Macs

From legendary Apple guru, Scott Knaster, find out all the coolest hacks, tweaks, and mods that you can make to your Mac®, iPod® or even to Mac OS® X Tiger™



EXTREME TECH™

Available wherever books are sold.

 **WILEY**
Now you know.

Wiley and the Wiley logo are trademarks of John Wiley & Sons, Inc. and/or its affiliates. The ExtremeTech logo is a trademark of Ziff Davis Publishing Holdings, Inc. Used under license. All other trademarks are the property of their respective owners.

BACK TO THE FUTURE, PART II

DEVELOPING APPLICATIONS WITH THE QUICKTIME FOR COCOA KIT

In the previous QuickTime Toolkit article (“Back to the Future” in MacTech, May 2005), we began looking at Apple’s new Cocoa framework for displaying and modifying QuickTime movies, called the QuickTime For Cocoa Kit or the QTKit. We saw how to open and display a movie in a window of a Cocoa document-based application; we also saw how to manipulate movies using command-line tools, which would be quite useful for batch processing via shell scripts or other scripting mechanisms.

Introduction

In this article, we’ll continue our investigation of the QTKit framework, which is available in QuickTime 7 on both Panther and Tiger. We’ll see how to extend the sample application KitEez that we began developing in the previous article to support the complete set of standard document behaviors (Save, Save As, Revert, and so forth). Toward the end of this article, we’ll see how to restrict the file types displayed in the file-opening dialog box. Then in the next article, we’ll wrap up our tour of the QTKit by looking at some advanced uses of its classes and methods.

Document Behaviors

So far, our sample application KitEez is able to open any kind of file that QuickTime can handle — standard movies files, Flash files, MP3 files, MPEG-4 files, AIFF files, text files, and so forth — and

display the movie data in a Cocoa document window. Figure 1 shows a typical KitEez document window. As you can see, the movie and the movie controller bar completely fill the content region of the window. As you can also see from the hourglass shape of the thumb in the movie controller bar, the movie is editable using the items in the Edit menu or their standard keyboard shortcuts.

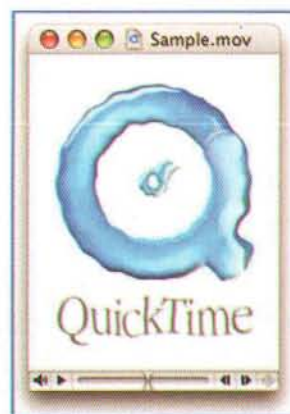


Figure 1. A window that contains a QTMovieView

Because movies opened using KitEez are editable, we'll want to support all the standard document behaviors: Undo/Redo, Save, Save As, Revert to the last saved version, and so forth. If you have developed Cocoa document-based applications before, then you'll know that Cocoa provides a sophisticated document architecture that tracks changes to a document and responds appropriately to user actions. For instance, if the user tries to close an edited but unsaved document window, Cocoa will display a sheet that prompts the user to save or discard those changes. The Cocoa document architecture, however, has no knowledge of what it means to save a QuickTime movie, so we'll need to override a few `NSDocument` methods to add that and other standard document behaviors to our sample application.

Supporting Editing Operations

It's actually trivially easy to support the standard undo and redo mechanism for edits made to a QuickTime movie that is displayed in a `QTMovieView` in a Cocoa document window. All we need to do is make the associated `QTMovie` object editable, and we saw how to do this in the previous article with this single line of code:

```
[movie setAttribute:[NSNumber numberWithInt:YES]
                    forKey:QTMovieEditableAttribute];
```

Once we've executed this line of code, the movie can be edited using `QTMovieView` methods like `cut`, `copy`, `paste`, `trim`, and the like. Indeed, the default behavior of the Edit menu items is to issue these IB actions, so we need to write exactly 1 line of code (shown above) to have a full-fledged movie editing application.

QTKit provides the standard multiple-level undo and redo processing. A user can cut, copy, and paste to his or her heart's content and then undo all those editing operations. Also, QTKit automatically provides the standard drag-and-drop editing operations. A user can drag a movie segment out of a movie window and into any other object that can accept dragged movie data. Figure 2 shows the drag image being dragged out of a movie window.

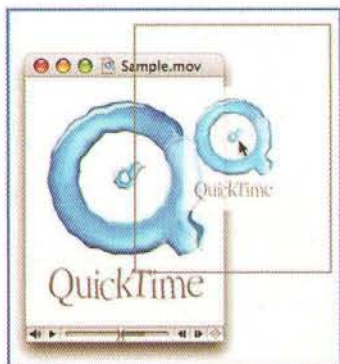


Figure 2. A dragged movie segment

The performance of the dragging in QTKit-based applications is vastly improved over that provided by the standard movie controller. In some benchmarks, the creation of the drag image alone was over 3000 times faster in QTKit applications than in applications using the movie controller's drag-and-drop capability.

Saving an Edited Movie

When the user selects the Save item in the File menu, the first responder's `saveDocument:` method is executed. The `NSDocument` implementation of this method checks to see whether a file is already associated with the document; if not, it displays the file-saving sheet to elicit a filename and location from the user. This will never actually happen in KitEez, since we only ever open movies from existing files. But it might happen that an edited movie cannot be saved into the file it was opened from. For instance, KitEez is able to open MP3 files as QuickTime movies (transparently invoking its MP3 import component), which are then fully editable. But QuickTime does not provide an MP3 export component, so KitEez is unable to save the edited data back into the original MP3 file. In that case, we'd want to display the file-saving sheet so that the user can save the data as a QuickTime movie file (.mov).

The `QTMovie` class provides a method that we can use to determine whether a movie object can be saved into its associated file, `canUpdateMovieFile`. Listing 1 shows our override of the `saveDocument:` method.

Listing 1: Saving a document

```
- (IBAction)saveDocument:(id)sender
{
    if ([[movieView movie] canUpdateMovieFile])
        [super saveDocument:sender];
    else
        [super saveDocumentAs:sender];
}
```

This is simple enough: if the movie can be saved into the file it was opened from, then we just call `NSDocument`'s `saveDocument:` method; otherwise we call its `saveDocumentAs:` method.

`NSDocument`'s `saveDocument:` and `saveDocumentAs:` methods internally call `writeWithBackupToFile:ofType:saveOperation:` to actually write the document data into the associated file. We can override that method to save the edited movie data into the original movie file or into the newly-selected destination file. Listing 2 shows our override method.

Listing 2: Writing out the movie data

```
- (BOOL)writeWithBackupToFile:(NSString *)fileName
ofType:(NSString *)documentTypeName
```



```

    saveOperation:(NSSaveOperationType)saveOperationType
{
    BOOL success = NO;

    if (saveOperationType == NSSaveOperation)
        success = [[movieView movie] updateMovieFile];
    else if (saveOperationType == NSSaveAsOperation) {
        success = [[movieView movie] writeToFile:fileName
                                                withAttributes:nil];
        if (success) {
            QTMovie *newMovie = [QTMovie movieWithFile:fileName
error:nil];

            [movieView setMovie:newMovie];
            [self initializeMovieWindow];
        }
    }

    return success;
}

```

When the requested operation is `NSSaveOperation`, we simply call the `updateMovieFile` method to update the movie atom in the movie file. When the requested operation is `NSSaveAsOperation`, we need to do a little more work. As you can see, we first call the `QTMovie` method `writeToFile:withAttributes:` with a `nil` dictionary of attributes. This has the effect of writing *only* the movie atom into the destination file. What we end up with in this case is a reference movie, that is, a movie whose media data is located in some other file. If instead we wanted a self-contained movie file, we could call `writeToFile:withAttributes:` with a dictionary that indicates that we want the movie to be flattened into the destination file, like this:

```

success = [[movieView movie] writeToFile:fileName
                                withAttributes:[NSDictionary
dictionaryWithObject:[NSNumber numberWithInt:YES]
forKey:QTMovieFlatten]];

```

I'll leave it as an easy exercise for the reader to add an accessory view to the save panel that allows the user to determine whether a movie is saved as a reference movie or as a self-contained movie, like the one displayed by the QuickTime Player application (Figure 3).

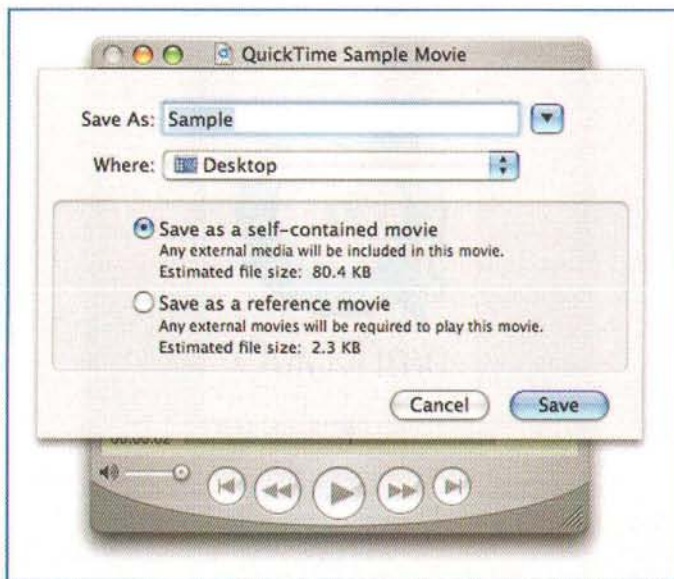


Figure 3. An accessory view in a save panel

As you know, the standard behavior when performing a Save As operation is to replace the movie in the movie view with the movie in the new destination file. So, once we've called `writeToFile:withAttributes:`, we then need to open the movie in the new file and set it as the movie displayed in the movie view. Then we need to perform any movie configuration that must happen when a movie is opened. As you can see, we call the method `initializeMovieWindow`, defined in Listing 3. This code is lifted wholesale from the `windowControllerDidLoadNib:` method we considered in the previous article on QTKit.

Listing 3: Initializing a movie and its window

```

- (void)initializeMovieWindow
{
    QTMovie *movie = [movieView movie];

    // make the movie editable

    [movie setAttribute:[NSNumber numberWithInt:YES]
forKey:QTMovieEditableAttribute];

    // use the QTKit's resize indicator, not NSWindow's

    [[movieView window] setShowsResizeIndicator:NO];
    [movieView setShowsResizeIndicator:YES];

    // add a listener for size-changed notifications

    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(boundsDidChange:)
name:QTMovieSizeDidChangeNotification object:movie];
}

```

It's perhaps worth mentioning that the `writeWithBackupToFile:ofType:saveOperation:` method is deprecated in Mac OS X version 10.4 and later. Since however we want our application to run under all environments supporting QTKit, including Mac OS X version 10.3.9, we'll stick with this method.

Reverting to the Last Saved Version of a Movie

There remains just one document behavior for us to implement, namely reverting to the last saved version of a movie file. When the user selects the Revert item in the File menu, the Cocoa document handling code displays the sheet shown in Figure 4.

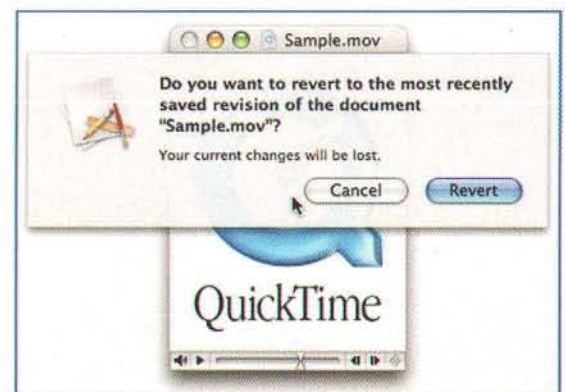


Figure 4. The Revert sheet

If the user presses the Revert button, the `revertDocumentToSaved:` method in the first responder is invoked. We won't override that method, so that method in the `NSDocument` class will be called. Rather, we'll override the `revertToSavedFromFile:ofType:` method, as shown in Listing 4.

Listing 4: Reverting to a saved movie file

```
- (BOOL)revertToSavedFromFile:(NSString *)fileName
    ofType:(NSString *)type
{
    QTMovie *newMovie = [QTMovie movieWithFile:fileName
error:nil];

    [movieView setMovie:newMovie];
    [self initializeMovieWindow];

    return YES;
}
```

As in the Save As case shown in Listing 3, we open the specified file and assign it to the movie view; then we perform the required initialization of the movie and movie window. Also, we return the value YES to make sure that the change count of the document is cleared.

So, we have successfully implemented a reasonably complete set of document-related behaviors in our sample application KitEz, using just a very few QTMovie methods: `canUpdateMovieFile`, `updateMovieFile`, and `writeToFile:withAttributes:`. At no point did we need to step

outside the methods provided by QTKit into the underlying Carbon APIs. This stands in stark contrast to the work we needed to do in our earlier Cocoa movie-playing application MooVeez, which relied on the now-deprecated Cocoa classes `NSMovie` and `NSMovieView` for its QuickTime support. I invite you to take a second look at the article describing those classes ("The Cocoanuts" in *MacTech*, December 2002); by comparing the Carbon-laced methods we had to write there with the purely Cocoa-based methods we've seen here, you will get a very good appreciation of how big an advancement the QTKit is over those earlier classes.

Openable File Types

In the previous article introducing QTKit, we saw that we can very easily configure the file-opening dialog box to list all files openable by QuickTime by overriding the `panel:shouldShowFilename:` method and calling the `canInitWithFile:` class method. That method returns YES just in case a specified file can be used to initialize a QTMovie object. For certain purposes, however, we might want to restrict the types of files that the user can open to some more limited set. For instance, as you probably know, QuickTime can open text files with its text movie importer — which converts the text into a series of frames in a text track. (See "Word is Out" in *MacTech*, November 2000 for more information on QuickTime's text-handling capabilities.)

OPENBASE
**SUMMIT
2005**
DEVELOPERS
CONFERENCE

SEPTEMBER 14-18 2005 • THE MOUNT WASHINGTON RESORT • BRETTON WOODS, NH

Register
Now

Build Better Software

with Cocoa, WebObjects, REALbasic, PHP, Java, and Core Data

SUMMIT 2005 offers over 30 technical sessions lead by industry experts that will increase your productivity.

Visit www.openbase.com/summit2005 for details.

Use the latest tools and techniques to...

- build powerful VOIP applications
- synchronize with QuickBooks
- develop more effective solutions
- ...and more.

But it's doubtful that most applications that can open and display QuickTime movies would want to allow text files to be opened by the user. Ditto for HTML files. Ditto for PDF files.

QTKit provides an easy way to allow the user to open only the kinds of files that would normally be considered as media files and hence openable by a QuickTime application. Instead of using the `canInitWithFile:` method, we can use the `movieFileTypes:` method. This too is a class method, so you don't need to have an existing `QTMovie` object in order to call it.

The `movieFileTypes:` method returns an `NSArray` of file types and file extensions that can be opened by QuickTime. This method takes one parameter, which indicates the kinds of file types that you want to be included in the array. Currently these flags are defined for specifying file types:

```
typedef enum {
    QTIncludeStillImageTypes      = 1 << 0,
    QTIncludeTranslatableTypes    = 1 << 1,
    QTIncludeAggressiveTypes      = 1 << 2,
    QTIncludeCommonTypes          = 0,
    QTIncludeAllTypes             = 0xffff
} QTMovieFileTypeOptions;
```

Passing the value `QTIncludeCommonTypes` indicates that you want the array to contain only those file types and extensions that are commonly thought of as openable by QuickTime; this includes the extensions `.mov` and `.mqv`, as well as any file types that can be imported in place by QuickTime. (Recall that to be able to import a file *in place* is to be able to import the file without having to create a new file to hold the imported data.)

The `QTIncludeStillImageTypes` flag indicates that you want the array to include, in addition to the common file types, all types of files that can be opened by QuickTime using one of its available graphics importers. The `QTIncludeTranslatableTypes` flag indicates that you want the array to include also those file types that can be opened by QuickTime using one of its available movie importers, whether or not the file data can be imported in place; this array excludes however any file types that would require an aggressive importer. (An aggressive importer is one that can handle file types like text or HTML that would not normally be considered openable by QuickTime.) Finally, the `QTIncludeAggressiveTypes` flag indicates that you want the array to include file types that require an aggressive importer.

Obviously, the `QTIncludeAllTypes` flag sets all the bits in the options parameter and requests that the returned array include file types and file extensions for all files that QuickTime can open, using any available graphics or movie importer. Passing the array associated with the `QTIncludeAllTypes` flag to `NSOpenPanel`'s `runModalForTypes:` method is essentially identical to using the `canInitWithFile:` method inside our `panel:shouldShowFilename:` override method.

It's important to realize that, for instance, the array returned when specifying the `QTIncludeStillImageTypes` flag will include all file types and extensions for still image files *as well as* all file types and extensions for the common movie types. That is to say, that array does not contain *only* the file types and extensions for still image types. If you wanted an array like that, you would need to

use Component Manager functions to iterate through all installed components and select only those of type `GraphicsImporterComponentType`. Listing 5 shows how you might do that.

Listing 5: Finding still image types

```
fileTypes = [NSMutableArray array];
ComponentDescription findCD = {0, 0, 0, 0, 0};
ComponentDescription infoCD = {0, 0, 0, 0, 0};
Component comp = NULL;
OSErr err = noErr;

findCD.componentType = GraphicsImporterComponentType;
findCD.componentSubType = 0;
findCD.componentManufacturer = 0;
findCD.componentFlags = 0;
findCD.componentFlagsMask = cmpIsMissing |
                             graphicsExporterIsBaseExporter;

while (comp = FindNextComponent(comp, &findCD)) {
    err = GetComponentInfo(comp, &infoCD, nil, nil, nil);
    if (err == noErr) {
        if (infoCD.componentFlags &
            movieImportSubTypeIsFileExtension)
            [fileTypes addObject:[NSString
                stringWithCString:(char *)&infoCD.componentSubType
                length:sizeof(OSType)]
                stringByTrimmingCharactersInSet:[NSCharacterSet
                    whitespaceCharacterSet]] lowercaseString];
        else
            [fileTypes addObject:[NSString
                stringWithFormat:@"%s", [NSString
                    stringWithCString:(char *)&infoCD.componentSubType
                    length:sizeof(OSType)]]];
    }
}
```

Conclusion

In this article, we've added the standard document-handling behaviors to our KitEz sample application. We've seen how to use QTKit methods to support saving an edited movie into its original file or into a new file, and we've seen how to revert to the last saved version of a movie file. KitEz is now a reasonably complete multi-document movie opening and editing application.

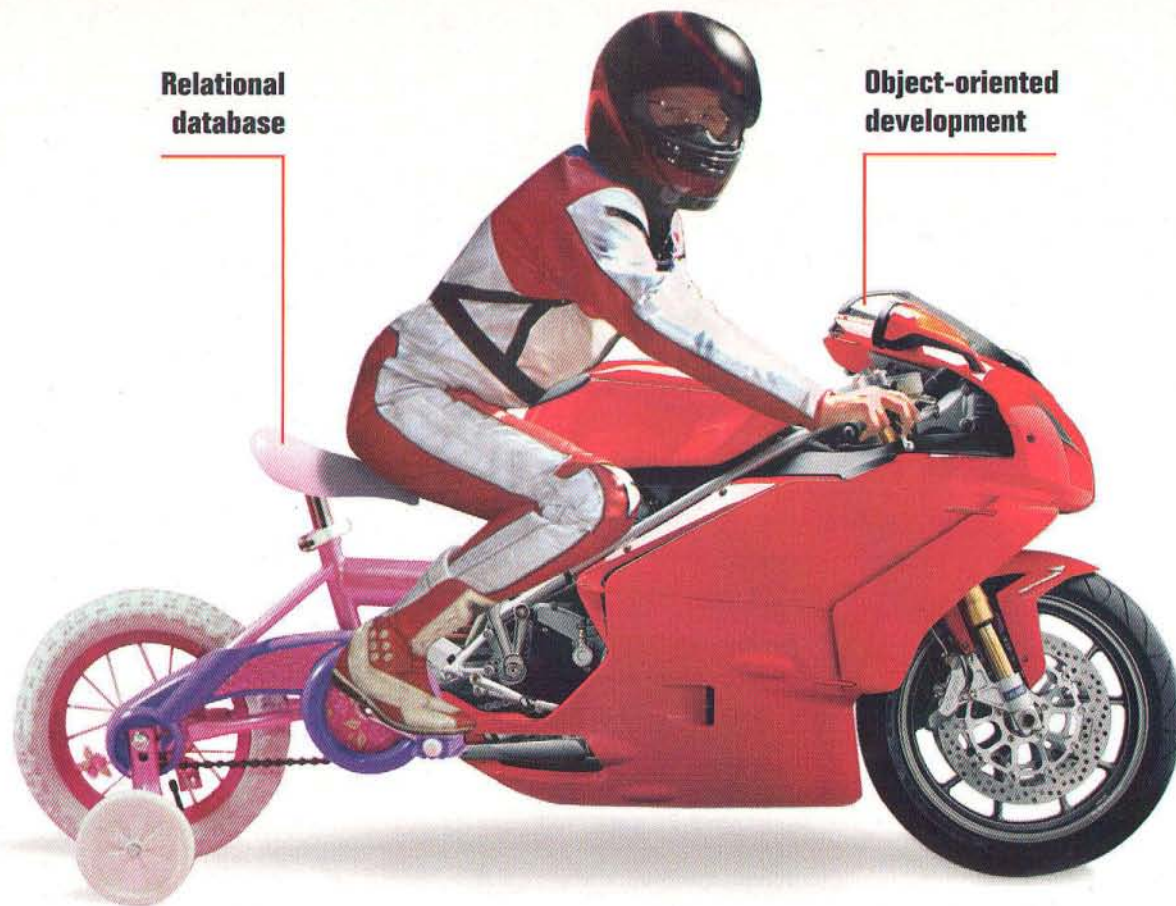
In the next article, we'll take a look at some of the more advanced operations we can use the QTKit framework to perform. We'll see how to add images to an existing QuickTime movie, work with notifications and delegates, and execute QTKit methods on a secondary thread.

MM

About The Author

Tim Monroe is a member of the QuickTime engineering team at Apple. You can contact him at monroe@mactech.com. The views expressed here are not necessarily shared by his employer.





A BETTER DATABASE CAN SPEED UP YOUR DEVELOPMENT CYCLE

If your relational database isn't a good match for your object-oriented development, you need a new database.

Caché, the *post-relational* database from InterSystems, combines high-performance SQL for faster queries and an advanced object database for rapidly storing and accessing objects. With Caché, no mapping is required between object and relational views of data. That means huge savings in both development and processing time.

Applications built on Caché are massively scalable and lightning fast. They require little or no database administration.

More than just a database system, Caché incorporates a powerful Web application develop-

ment environment that dramatically reduces the time to build and modify applications.

Caché is so reliable, it's the world's leading database in healthcare – and it powers enterprise applications in financial services, government and many other sectors. With its high reliability, high performance and low maintenance, Caché delivers your vision of a better database.

We are InterSystems, a specialist in data management technology for over twenty-six years. We provide 24x7 support to four million users in 88 countries. Caché is available for Windows, OpenVMS, MAC OS X, Linux, and major UNIX platforms – and it is deployed on systems ranging from two to over 50,000 simultaneous users.



Try a better database. For free.

Download a free, fully functional, non-expiring copy of Caché or request it on CD at www.InterSystems.com/match3



DevDepot has it all!

Get More out of your Mac!

DevDepot sells the tools, toys and technology to put more muscle into your Mac. Visit our online store today for special offers and great new products. www.devdepot.com

Television is about to get a whole lot more interesting.

EyeTV 200

The future of Television

Watch TV on your Mac!

Pause live tv! Record, edit
and archive!



NOW ONLY
\$324⁹⁹

go
digital

Convert your analog
video tapes to
digital in realtime,
then burn them
to DVD! *

Watch TV on your Mac

EyeTV features a 124-channel cable-ready analog tuner and DVD quality MPEG-2 video encoder.

Record TV on your Mac and Archive to Disk

Collect and organize your favorite shows to watch whenever you want, then update video content to DVD.* (Toast 6 Titanium required).

Don't miss a thing

Use EyeTV's Electronic Program Guide to find exactly what you are looking for, and program EyeTV to record it. **Program EyeTV from anywhere via the Internet.**

Features and Benefits

- Record TV on Your Mac
- Edit Out Unwanted Content
- Archive Recorded TV To DVD*
- MPEG-2 Video Encoding
- Digitize Analog Video
- The Speed And Power Of Firewire

NEW!

New Export Functionality

EyeTV now lets you export to iMovie®, iDVD® and DVD Studio Pro®, making it easier to create professional quality recordings.

from **elgato**

More Great Products!

Buy Today and Save!

SYNCBOX

Portable USB Data Copier

ONLY
\$43⁹⁹

Transfer data from your USB devices with the touch of a button.

*** Never run out of memory space again!**

- No computer required, 100% portable
- Sleek and compact design
- Easy to use, one button operation
- Complies with USB 1.1 specifications
- Supports both USB 1.1 and USB 2.0
- Uses 3 AAA alkaline batteries (not included)

DIGITAL CAMERAS



FLASH DRIVES



EASILY GET DATA FROM:

- Storage Devices
- MP3 Players
- Digital Cameras
- Card Readers
- Flash Drives
- AND MORE!



FITS IN YOUR HAND!

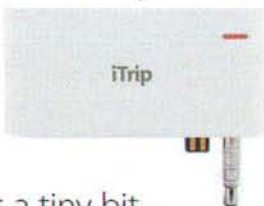
from **macally™**

iTripmini

FM Transmitter

The iTrip mini was designed exclusively for the iPod mini.

Listen in your car!



NO BATTERIES NEEDED!

The iTrip mini only needs a tiny bit of power that it gets directly from your iPod mini.

ONLY
\$39⁹⁹



Its form matches all the curves of the iPod mini, and sounds even sweeter!

a Griffin Technology

iMic

USB Audio Interface

The iMic is a must-have device for people who are serious about high quality audio.

Connect virtually any sound device to your iBook, PowerBook, PowerMac or other Mac or PC with a USB port.

iMic SUPPORTS:

- Line Level Microphones
- Mic Level Microphones
- Multimedia Devices
- Headsets
- Communications Devices



ONLY
\$34⁹⁹

from Griffin Technology



DevDepot is not responsible for typographical errors. Offers subject to change at any time. © 1984-2004 Developer Depot, Inc. Some material copyright of their respective holders. All Rights Reserved. Developer Depot, Inc. is a division of Allume Systems, Inc. located in California.

www.devdepot.com

Mac OS X Programming

Save Our Screens - 101 -

How to write a Mac OS X screen saver, part 1.

By David Hill

Introduction

Everybody out there who has used a computer for more than a few minutes knows what screen savers are and there have been many screen savers written for the Macintosh and other computers. Some were single-purpose and provided only one animation but did it very well. Many others took a different approach and provided a generic engine that made use of plugin modules. With Mac OS X, Apple has included a Cocoa plugin based engine controlled via a System Preferences pane (Figure 1). The engine provides screen locking, window management, multiple screen support, and other features, freeing up the plugin developer to worry about the part that differentiates their product, the animation. A minimal Mac OS X screen saver module needs to worry about nothing more than drawing the next frame of its animation when requested to by the screen saver engine.

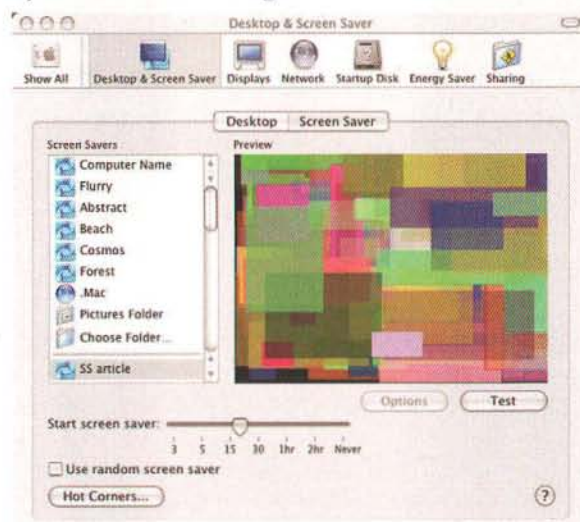


Figure 1. Screen shot of the Panther Screen Saver Preferences pane

Writing screen savers is easier than ever with Mac OS X and in this article I'm going to start by showing you how to write a simple module. We'll begin with the basics of screen saver construction in Xcode and work our way up to handling events, providing a configuration sheet, and changing display resolution. Along the way, we'll talk about different drawing APIs, debugging tips, random numbers, and user preferences. I'll also cover some tips and tricks as well as point out some trouble spots you'll need to watch out for.

For the purposes of this article, I'm going to assume that you have Mac OS X and the developer tools installed if you want to follow along. You'll need some development experience and a basic

knowledge of C. The interface for the screen saver plugins is in Objective-C but, since Objective-C is a close cousin of C, most C/C++ and Java developers should be able to follow along without too much trouble. For those who would like to learn more about Objective-C, Apple provides some excellent documentation on their web site at <http://developer.apple.com/documentation/Cocoa/ObjectiveCLanguage-date.html>. I'll also give all of my directions and screen shots with respect to Mac OS X version 10.3 (Panther) and Xcode so those of you running different versions of Mac OS X will need to mentally map from 10.3 to your OS version and development environment.

Basics

The `ScreenSaver.framework` provides two screen saver-specific classes that you'll need (`ScreenSaverView` and `ScreenSaverDefaults`) as well as a few utility functions for generating random numbers and centering rectangles. The most important of the two classes is `ScreenSaverView`, a subclass of `NSView`. The `NSView` class is responsible for drawing to the screen and handling user input. The `ScreenSaverView` class derives from `NSView` and adds support for animation plus a few other screen saver specific methods.

One part of screen saver development that can be confusing to developers is the question of where to do their drawing. If you look at the `ScreenSaverView` header, you'll notice that the `ScreenSaverView` class provides two promising methods: `animateOneFrame` and `drawRect:`. If you're familiar with Cocoa's view system, you'll recognize `drawRect:` as the standard method in which views do their drawing and it turns out that this is best place for your screen saver to do its drawing as well. This is particularly true for more advanced screen savers that make use of subviews to draw with OpenGL or QuickTime and we'll cover subviews in a later section. The reason to draw in `drawRect:` is that Cocoa will call `drawRect:` at various points in the life of your screen saver and you must be ready to handle those requests. For that reason, the recommended coding convention is to use `animateOneFrame` to update your animation state and then tell Cocoa that your view needs redrawing. This redraw request will cause Cocoa to call `drawRect:` where your screen saver will completely redraw the view to reflect its current state. In most screen savers, the `drawRect:` method erases the background, filling it with black or a background image, and then draws the objects that appear in the view.

However, there is one type of screen saver that needs to handle drawing slightly differently. Some simple screen savers accumulate content via repeated drawing into the view. By their very nature, they can't redraw the current state of the view because they don't keep track of what they've already drawn. In order to accumulate content, these simple screen savers must only erase the background of the view the first time they're asked to draw. The basic screen saver example I'll be using for the first part of the article falls into this stateless category so I'll

cover the extra bits of code you'll need to write an accumulating screen saver as we go along.

A typical screen saver will subclass `ScreenSaverView`, override a few important methods, add a few more of its own, and let the screen saver framework and engine handle the rest. The Xcode screen saver template provides stub implementations for 7 key methods but for a basic screen saver we'll only need to change the `-drawRect:` and `-animateOneFrame` methods, override two other superclass methods, and declare a new member variable. The rest of the stub code will suffice for now.

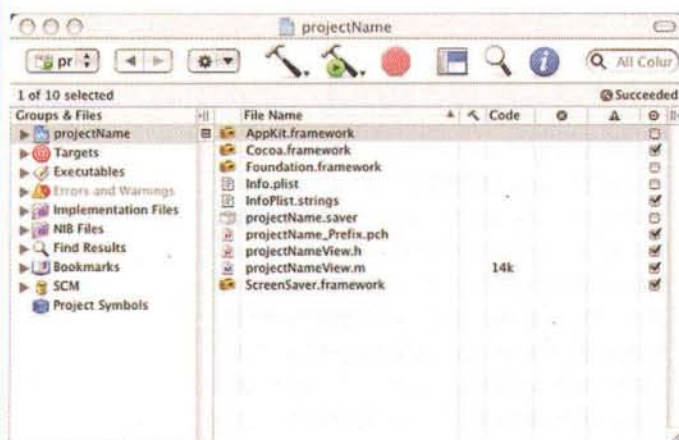


Figure 2. Initial screen saver project window

First, bring up Xcode and start a new screen saver project if you haven't already. You'll find the template called `Screen Saver` in the `Standard Apple Plug-ins` list. Give the new project a creative name. I've called my project `projectName` and so Xcode has created a few files for me with appropriate names. See Figure 2 for a look at the initial project. The most important one is called `projectNameView.m`. From this point on, I'll assume that your project is called `projectName` so open up your project and follow along as we add some simple drawing code.

1. Declare a new member variable in the `projectNameView.h` file with this code:

```
@interface projectNameView: ScreenSaverView {
    // keep track of whether or not drawRect: should erase
    the background
    BOOL mDrawBackground;
}
```

2. Add a new method to `projectNameView.m` called `viewDidMoveToWindow:`

```
-(void)viewDidMoveToWindow {
    // this NSView method is called when our screen saver
    view is added to its window
    // we'll use this signal to tell drawRect: to erase the
    background
    mDrawBackground = YES;
}
```

3. Replace the `drawRect:` method in the `projectNameView.m` file with this code:


```

- (void)drawRect:(NSRect)rect {
    if ( mDrawBackground ) {
        //draw background after view is installed in a window
        for the first time
        [[NSColor colorWithDeviceRed: 0.0 green: 0.0
                                     blue: 0.0 alpha: 1.0] set];
        [NSBezierPath fillRect: [self bounds]];
        mDrawBackground = NO;
    }

    NSRect viewBounds = [self bounds];
    float startingX = SSRandomFloatBetween(
        NSMinX( viewBounds ), NSMaxX( viewBounds ) );
    float startingY = SSRandomFloatBetween(
        NSMinY( viewBounds ), NSMaxY( viewBounds ) );
    float width = SSRandomFloatBetween(
        NSWidth(viewBounds)/20, NSWidth(viewBounds)/2);
    float height = SSRandomFloatBetween(
        NSHeight(viewBounds)/20, NSHeight(viewBounds)/2);
    NSRect rectToFill = NSMakeRect( startingX, startingY,
                                    width, height );

    float red = SSRandomFloatBetween(0.0, 1.0);
    float green = SSRandomFloatBetween(0.0, 1.0);
    float blue = SSRandomFloatBetween(0.0, 1.0);
    float alpha = SSRandomFloatBetween(0.0, 1.0);
    [[NSColor colorWithDeviceRed: red green: green
                               blue: blue alpha: alpha] set];
    [NSBezierPath fillRect: rectToFill];
}

```

4. Add a new method to projectNameView.m called isOpaque:

```

- (BOOL)isOpaque {
    // this keeps Cocoa from unnecessarily redrawing our
    // superview
    return YES;
}

```

5. Change the animateOneFrame method in projectNameView.m to look like this:

```

- (void)animateOneFrame {
    //request that our view be redrawn (causes Cocoa to call
    //drawRect:)
    [self setNeedsDisplay: YES];
}

```

Build your screen saver and correct any errors that Xcode finds. Once your project has built successfully, you'll find the built version of the screen saver in your project's build directory. Look for the file named `projectName.saver`.

Now that you've built your screen saver, I need to take a moment to explain how to get the system to recognize and display it. Screen savers must be installed into one of two places on your system in order for the screen saver engine to load and run them. If you'd like to make your screen saver available to all users on your system, use the Finder to copy it into `/Library/Screen Savers/` but note that you'll need administrative privileges for the copy operation. If, however, you only want the screen saver to be available to the current user, copy the plugin into `~/Library/Screen Savers/`. Once you've placed the screen saver into one of the two directories, open System Preferences and choose the Screen Saver tab within the Desktop and Screen Savers pane. Your new screen saver should show up in the list on the left-hand side. Select it and click the **Test** button to take your plugin for a spin.

If everything went according to plan (does it ever?), the screen saver engine should take over the display, load your

module, and start calling `-animateOneFrame` periodically to update the screen. If that's not what happened on your machine, here is a checklist of possible trouble spots.

1. Screen saver didn't show up in the list verify `.saver` extension and installation location, relaunch System Preferences
2. Screen saver shows up in list but doesn't preview verify subview origin (if you're using one) and see debugging tips
3. Screen saver previews but doesn't test verify subview origin and drawing coordinates
4. Preview draws in the wrong place verify subview origin
5. Screen saver crashes check console for OS and ScreenSaverEngine messages and see debugging tips
6. Weird System Preferences pane behavior check for possible symbol name conflict with another Objective-C bundle
7. Screen saver repeatedly draws a single random rectangle over a pin-striped background make sure you spelled `isOpaque` correctly

Did it work? Great! You'll notice that the end result isn't too bad for a few lines of code. The frameworks do quite a bit of work for you. As a quick example of the extra power available to your screen saver, add the following code right before the call to `NSBezierPath` at the end of `drawRect:` that fills the random rectangle.

```

NSAffineTransform* rotation =
    [NSAffineTransform transform];
float degrees = SSRandomFloatBetween( 0.0, 360.0 );
[rotation rotateByDegrees: degrees];
[rotation concat];

```

Build your saver, copy it over, and try it out. Did the screen saver behavior change? If not, you've probably forgotten to copy the updated screen saver plugin into the Screen Savers folder. Clearly, having to copy the updated build over every time you make a change will get old during development. The simplest way to fix this is to add a **Copy Files** build phase to your project that copies the finished screen saver from the build directory to `/Library/Screen Savers/` or `~/Library/Screen Savers/` once the build is complete. Open your project window, select your screen saver in the **Targets** list, and select **New Build Phase** -> **New Copy Files Build Phase** from the **Project** menu. See Figure 3. Once you've added the new build phase, you must set two additional parameters: the source files and the destination path.

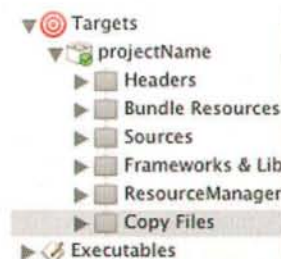


Figure 3. Targets list with a Copy Files Build Phase

If the Copy Files Info window isn't already open, select the Copy Files build phase and hit CMD-I to open the Copy Files Info window. Select Absolute Path from the Destination pop up menu and type the path to your desired installation location into the Path edit field. Figure 4 shows the Copy Files Info window with the correct settings. Note: there appears to be a bug (already filed) in the Copy Files build phase that prevents it from resolving the ~ (tilde) character so you can't just type in ~/Library/Screen Savers/ to copy your screen saver into the current user's folder. You'll need to type in the whole, explicit path to the user's Screen Savers directory if you want to install the screen saver for just the current user. If, on the other hand, you want the screen saver installed for all users on the machine, you can type /Library/Screen Savers/ into the Path box and the Copy Files build phase will work just fine. Coupled with a custom executable in Xcode, you'll be able to run and debug your screen saver directly from Xcode. See the debugging section below for more information on adding a custom executable to your project.

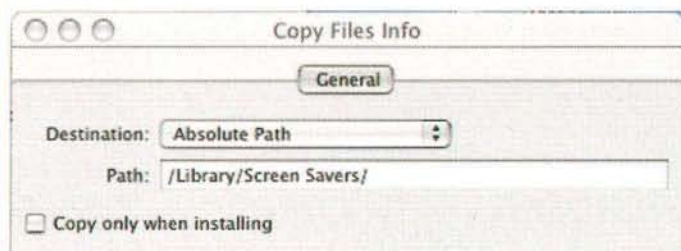


Figure 4. Copy Files Info window

To set the files to be copied, click the disclosure triangles to display the contents of both the Products group and the projectName target and then drag the projectName.saver item from the Products group to the Copy Files build phase. If you do it right, you should see a black horizontal line with a circle on the left end that indicates where the dragged item will go when you drop it. Make sure that the line is just under the Copy Files build phase item when you let go of the mouse button. See Figure 5 to see what I mean.

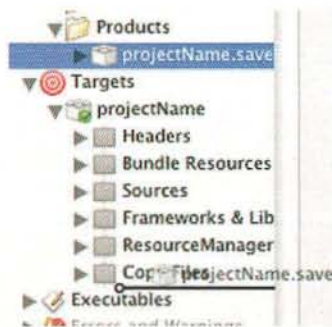


Figure 5. Setting up the Copy Files Build Phase

Another gotcha you may run into is that the Screen Savers preferences pane cannot unload your old saver and load in a new version when you make changes while you have the Screen Saver pane open in the System Preferences application. Either quit the

System Preferences application before making changes to your screen saver or, even better, add the Copy Files build phase and custom executable to your Xcode project which will allow you to run your screen saver from Xcode instead of the System Preferences application. Don't worry if you're still having trouble. I'll cover more debugging tips later on. In the next section, we'll take a closer look at the ScreenSaverView class and its methods.

ScreenSaverView Class

As I mentioned earlier, the ScreenSaverView class adds a few screen saver specific methods to the NSView class. Let's take a look at those new methods and see how they work together to save our screens. I've divided the list into two parts: the methods that you'll need to override and the methods that you'll merely need to call.

Methods to Override

```
-(id)initWithFrame:(NSRect)frame isPreview:(BOOL)isPreview
```

This is ScreenSaverView's designated initializer.

- subclass initializers must call this method (see references for more information)

The template calls [super initWithFrame:isPreview:] and sets the animation rate.

```
-(void)drawRect:(NSRect)rect
```

Use drawRect: to do your drawing.

Call [super drawRect:] to get ScreenSaverView's default black background.

```
-(void)animateOneFrame
```

Use animateOneFrame to update any per-frame state and invalidate your view.

```
-(void)startAnimation
```

This is your chance to set up resources and do any pre-processing necessary for animation.

You must at least call [super startAnimation] as shown in the standard template.

```
-(void)stopAnimation
```

This is your chance to free up any resources allocated in startAnimation.

You must at least call [super stopAnimation] as shown in the standard template.

```
-(BOOL)hasConfigureSheet
```

Tell the engine whether or not you're providing a configure sheet.

YES enables the Options button in the Desktop & Screen Savers System Preferences pane.

```
-(NSWindow*)configureSheet
```


If `hasConfigureSheet` returns YES, return the `NSWindow*` for your sheet here

```
+(BOOL)performGammaFade
```

Determines how the screen saver engine transitions to your screen saver

- return YES (default) for a gradual fade to black before your screen saver starts
- return NO to have your screen saver start immediately

Methods to Call

```
-(void)setAnimationTimeInterval:  
      (NSTimeInterval)timeInterval
```

`timeInterval` specifies the desired delay in seconds between frames.

The default time interval is 1/30.0, corresponding to 30 frames per second.

Zero polls as fast as possible while a negative number turns animation off.

```
-(BOOL)isAnimating
```

Check if your screen saver is currently between `startAnimation` and `stopAnimation`.

Useful for pausing calculations you don't want to perform while the screen saver is idle.

```
-(BOOL)isPreview
```

Call this method to see if you're being asked to draw the small preview.

Utility Functions in `ScreenSaverView.h`

```
SSRandomIntBetween
```

Generates a random integer in the closed interval `[a,b]`. I.E. the value returned will be one of `a, a+1, a+2, ..., b-1, b`.

```
SSRandomFloatBetween
```

Generates a random floating point value in the closed interval `[a,b]`

```
SSRandomPointForSizeWithinRect
```

Picks a point within the rectangle, leaving gaps on the right and bottom edges.

```
SSCenteredRectInRect
```

Returns a rectangle the size of the first rectangle but centered within the second rectangle.

Debugging Tips

Whether you're having trouble getting the basic screen saver up and running or you've moved on to more advanced

topics and are having problems, there are a number of debugging options and tricks available to help.

1. One of the best ways to improve your ability to debug a screen saver is to add the **Copy Files** build phase I mentioned above. Not having to copy newly built versions over after every change will drastically speed up your build-run-debug cycle. With the new build automatically available, you can build your screen saver and then immediately invoke it. You'll also want to disable the password protection option on your screen saver so that you don't have to type in your password every time. In Panther, that setting has moved to the **Security** pane in the **System Preferences** application.
2. Another good way to debug screen savers is to add a custom executable to your project that invokes the `ScreenSaverEngine` with the proper `-debug` and `-module` arguments. The `-debug` flag tells the `ScreenSaverEngine` to draw your screen saver behind all other windows rather than taking over the screen. This will allow you to run or debug your screen saver from within Xcode while still having access to your source files for debugging purposes. Note that unhandled events will still wake up your screen saver so you'll need to override the mouse and keyboard event methods in `NSResponder.h` and not call the implementation in the superclass. To add a custom executable, bring up your project window and select **Custom Executable** from the **Project** menu. In the subsequent **Assistant** dialog, name the custom executable anything you like and then set the path to `/System/Library/Frameworks/ScreenSaver.framework/Version s/A/Resources/ScreenSaverEngine.app`. This will add a new entry in the **Executables** group in your project. See Figure 6 for a look at the custom executable Assistant.

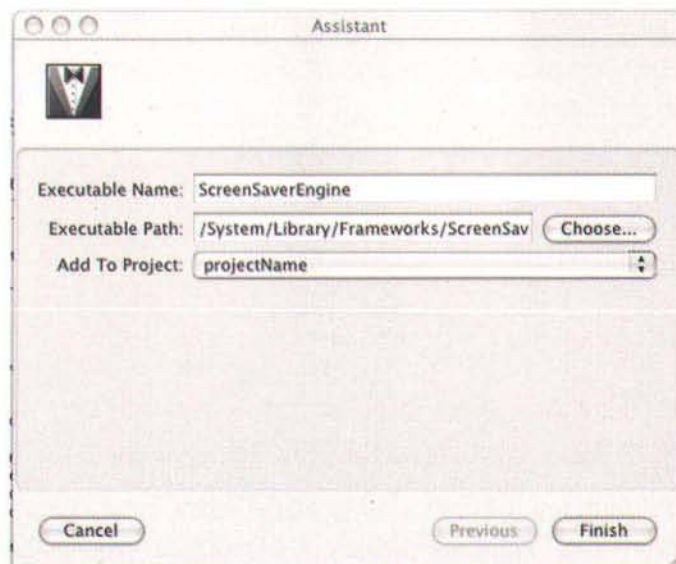


Figure 6. The Custom Executable Assistant window

Select that new entry and open its editor pane (or double click the entry to open the editor in a separate window). Click twice on the small round "plus" button directly below the Arguments section of the editor to add two new entries. Type in `-debug` for the first value and `-module "projectName"` (note that there is no `.saver` extension needed) for the second. This will tell the `ScreenSaverEngine` to run your module in debug mode regardless of the screen saver selected in System Preferences. Figure 7 shows the arguments in the custom executable editor. Build and run your project. Xcode should bring up the Run window and launch the `ScreenSaverEngine` application which will load and run your screen saver behind all of the other windows.

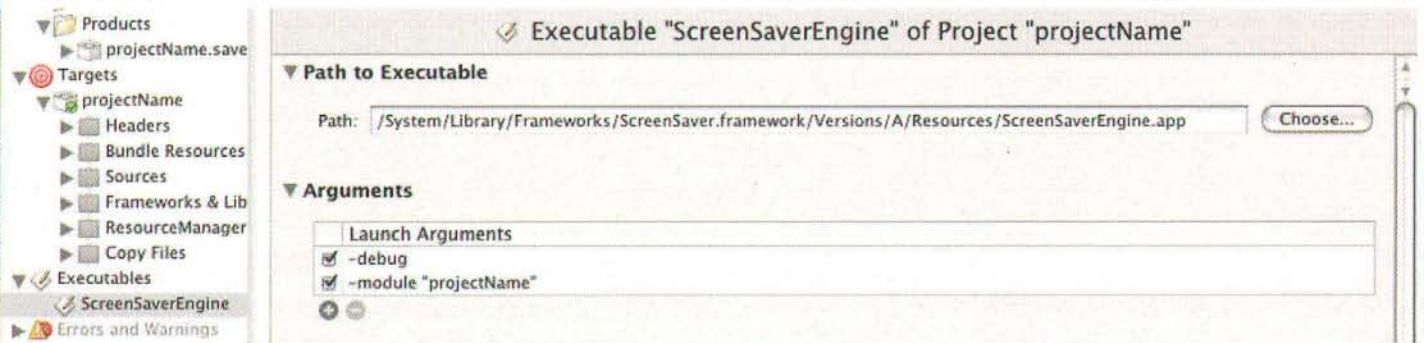


Figure 7. The Custom Executable editor

While this setup does make debugging much easier, I've run into a few trouble spots while writing this article that you should know about.

First, as with any code, if Xcode doesn't stop at your breakpoints make sure you're building and debugging your screen saver using the **Development** build style. The alternative, **Deployment**, strips the debugging symbols that the debugger needs to find its way around your code.

Second, your screen saver will be fighting Xcode for events and this can cause some confusion. When the screen saver first starts up, it will have the event focus and receive keyboard events. However, if you click in any of Xcode's windows, the screen saver will lose focus and keyboard events will start going to Xcode instead. If you were debugging a normal application, you'd simply click in one of its windows to restore keyboard focus to the application but screen savers aren't normal applications. The screen saver engine uses mouse and keyboard events as a signal to disengage the screen saver so even moving the mouse out of the debugger window into the screen saver window will cause the screen saver engine to shut down the screen saver and exit. The way

around this problem is to override a few more event handlers from `NSResponder.h`. At a minimum, you'll need to override `mouseDown:`, `mouseUp:`, `mouseEntered:`, and `mouseExited:` to do nothing so that you can move the mouse into the screen saver window and click to restore the keyboard focus without waking the screen saver. Don't forget to leave yourself some way of stopping the screen saver short of force quitting Xcode.

Third, even with the above code in place, you can run into areas where your screen saver state will get out of sync with the keyboard state. In particular, if you have breakpoints set on event handling methods like `mouseDown:`, `keyDown:`, or `flagsChanged:` you're going to have a bit of trouble because the Xcode debugger will eat the

corresponding "up" events when you hit those breakpoints. If your screen saver uses these events to toggle state like I'll show later on in the event handling example, you will need to either toggle the state variable in the debugger or disable the breakpoint, switch the keyboard focus back over to the screen saver window, and give it another down/up pair so that it catches the up event and restores the proper state.

4. In order to debug truly complicated event handling code, you may need to resort to debugging the screen saver remotely using `gdb` or just logging state and calling sequence information to `stdout` using `NSLog` or `printf`.
5. Cocoa has a rule that messages to `nil` do nothing and return `nil`. While this does help keep broken code limping along, it can make debugging particularly frustrating, especially when compounded by the fact that Cocoa lends itself to long lines of code with no intermediate variables. To see what I mean, take a look at this snippet of code from a section later in this article.

This code to load an image:

```
NSBundle* saverBundle = [NSBundle bundleForClass:
    [self class]];
NSString* imagePath = [saverBundle
    pathForResource:
        @"test image" ofType: @"JPG"];
UIImage* image = [[UIImage alloc]
    initWithContentsOfFile:
        imagePath];
```

could also be written like this:

```
UIImage* image = [[UIImage alloc]
    initWithContentsOfFile:
        [[NSBundle bundleForClass: [self
    class]]
        pathForResource: @"test image"
        ofType: @"JPG"]];
```

Although you'll often see code like the latter snippet, code written in this style can be very difficult to understand and debug. For your own sanity and that of anybody who has to read your code, please try to err on the side of the first snippet. It will make code much easier to read and debug, and the compiler should optimize out any intermediate variables so there's no penalty for using them.

6. Once you've got your screen saver up and running and you're just fine-tuning your graphics algorithms, you can build debugging help directly into the screen saver. Instead of relying on symbolic

debugging in Xcode or postmortem debugging via NSLog and the console.log file, add some code at the end of your `drawRect:` method to display the current state of your screen saver. Use `NSString` and the drawing functions in `NSStringDrawing.h` to draw status messages over the top of your animated content.

Summary

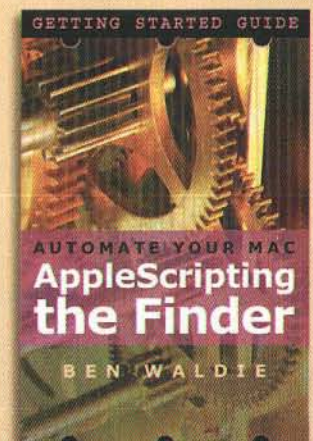
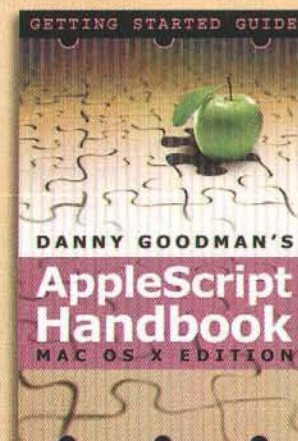
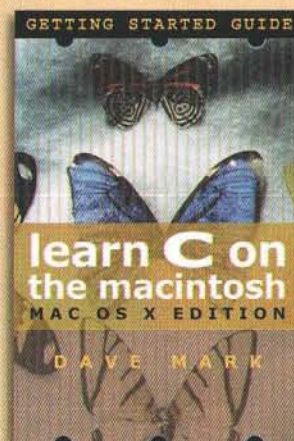
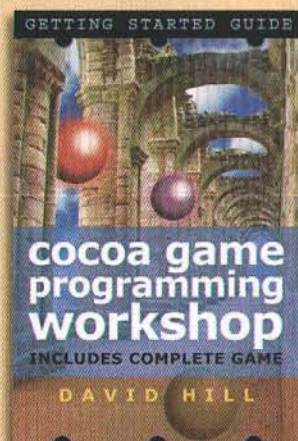
Now that we've worked through the creation of a simple screen saver, taken a closer look at the `ScreenSaverView` class and its methods, and covered some debugging tips, this is probably a good place to stop and take a break. In the next article, we'll look into some more interesting topics such as adding a simple configuration sheet, saving and loading those configuration options using `ScreenSaverDefaults`, and basic event handling. We'll also see how to use other drawing APIs within a screen saver.



About The Author

David Hill is a freelance writer living in College Station, Texas. In a former life, he worked in Apple's Developer Technical Support group helping developers print, draw, and write games. In his free time he dabbles in screen savers and other esoteric topics.

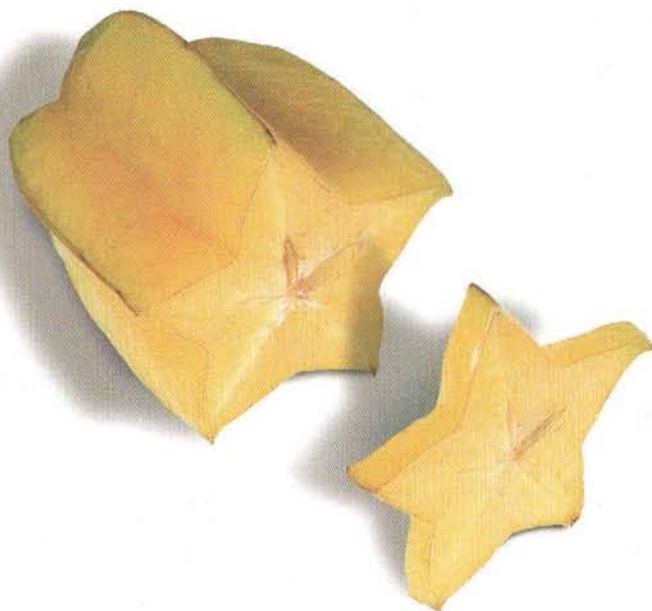
From our brain to your brain... SpiderWorks eBooks



For more information and to order online, visit

www.spiderworks.com

Shine.



Apache Bootcamp

Cocoa® Bootcamp

Core Bootcamp

PHP 5 Bootcamp

PostgreSQL Bootcamp

Python® Bootcamp

Core Mac OS® X Bootcamp

Get your dark sunglasses ready. Tell your boss to roll out the red carpet. Are you ready for your breakthrough role? Director Mark Dalrymple, author of *Core Mac OS® X* and *Unix Programming*, will take you from indies to a blockbuster in five days.

Sample Scenes:

Multithreading
Rendezvous™
System Daemons

Distributed Objects
Network Programming
Authentication & Authorization

Big Nerd Ranch offers intensive training classes for developers and administrators. Your expert instructor guides you through a rigorous week of learning. You leave ready to start developing (but with instructor support if you get stuck).

Big Nerd Ranch: Baby, we'll make you a star!

www.bignerdranch.com • 678.595.6773 • roundup@bignerdranch.com

LIFE WITH SQLITE

With Tiger, Apple has started using SQLite in many ways. Most notably, SQLite is the recommended store for Core Data applications.

What is SQLite?

A database server, for example Oracle or PostgreSQL, is a daemon that responds to requests from client applications. The clients make requests of the server using Structured Query Language (SQL) via some client library. The server takes care of the security, concurrency, and distribution issues. The server stores tables of data that can be indexed for quick random access.

SQLite is not a database server.

SQLite is an open-source C library written by Dr. Richard Hipp. It creates a file that holds tables of data that can be indexed for quick random access. You read and write to this file using SQL. The main advantage of SQLite over archiving is that the data can be read and updated incrementally. (An archive is read in one big piece, and must be completely rewritten if the resulting object graph is edited.) The performance and scalability of SQLite is excellent -- Dr. Hipp's mother has every right to be very proud of him.

You can learn all about SQLite at the website:

<http://www.sqlite.org/>

Looking at SQLite files

SQLite comes with a command-line tool called `sqlite3`. The tool is installed

automatically with Tiger. Using `sqlite3`, you can interactively inspect and edit any SQLite file. If you are curious about how Core Data structures the file, `sqlite3` is a great way to explore it. `sqlite3` has several non-SQL commands that start with a dot. `.schema` will show you the create statements for all the tables and indices in the file:

```
% sqlite3 test.eventsq
sqlite> .schema
CREATE TABLE ZLOCATION
  (Z_ENT INTEGER, Z_PK INTEGER PRIMARY KEY, Z_OPT INTEGER,
  ZDETAILDESCRIPTION VARCHAR, ZNAME VARCHAR );
CREATE TABLE ZPERSON
  (Z_ENT INTEGER, Z_PK INTEGER PRIMARY KEY, Z_OPT INTEGER,
  ZNOTES VARCHAR, ZFIRSTNAME VARCHAR, ZLASTNAME VARCHAR );
...
CREATE INDEX ZEVENT_ZOCCASION_INDEX ON ZEVENT (ZOCCASION);
sqlite> select * from zperson;
5 | 1 | 1 | Bride | Laura | Smith
5 | 2 | 1 | Groom | Craig | Adams
```

Watching the SQL executed by Core Data

This tip requires that you use some API that Apple has not yet exposed, and I would not use it in a shipping application. It is, however, a great way to understand exactly what Core Data is doing with SQLite. To log every SQL command sent to SQLite, execute the following code early in your application:

```
Class privateClass;
privateClass = NSStringFromClass(@"NSSQLConnection");
// The compiler will give a warning here
[privateClass setDebugEnabled:YES];
```

Using SQLite in non-Core Data applications

There are a couple of reasons why you might want to write an application that works with SQLite files, but doesn't use Core Data:

- 1) Backwards compatibility: Core Data is only available on Tiger.
- 2) Performance: While Core Data is very clever about which rows to fetch from a sqlite file, it fetches all the columns for those rows. If you know that you only need one or two columns, you may get a significant performance win from writing an explicit SELECT statement.

QuickLite is an Objective-C wrapper for SQLite by Tito Ciuro. It is an open source framework, and you can download it at:

<http://www.webbotech.com/>

One of the QuickLite example programs is SQLiteManagerX, a very useful Cocoa application for browsing and editing SQLite files.



Figure 1. SQLiteManagerX browsing a SQLite file

SQLite is not a database server

Before we end this, let us remind you that SQLite is not a database server. If more than one user is going to be accessing the data, you need a database server to take care of security, distribution, and concurrency issues. (There are, for example, many stories of companies trying to use Microsoft Access as a networked datastore. Most of the stories end with tears and a massive rewrite.) If your data is accessed by only one user, SQLite is an elegant and efficient solution.



Website Not Found By Clients HTTP 404 - Website not found

The website you are looking for can't be found by your clients. It may have been improperly marketed, had poor design, or didn't work. Regardless, it's not helping your business!

Your Options:

- Rent a chicken suit and stand on the corner handing out flyers
- Paint the company URL on your chest and face during a major sporting event
- Contact SharpNET Solutions internet marketing and web design specialists, watch your traffic and rankings increase, get great feedback from all your new customers.

HTTP 404 - Website needs SharpNET

Not Marketing Your Site?

If you are not marketing your website online, you may as well have a 404 error for a website.

Web Design
Internet Marketing
Consulting
Lead Generation
Multi-Media

1-877-583-8396
www.sharpsolutions.com



Serving From Home



Running A Mac Web Server With A Dynamic IPAddress

By Danny Swarzman

Introduction

With Mac OS X, the Macintosh has become an ideal platform for developing web applications. It includes an Apache server. All the services you want — FTP, PHP, Perl, MySQL, Tomcat, etc. — are either included with OS X or are available for free. You can test the client side on the same Macintosh that is running the server. You can edit web files with a regular text editor without an extra step to transfer them to a server.

Some accounts offered by Internet service providers don't offer all these services. Some accounts don't include access to the unix shell or don't allow you to view system log files— useful when you are debugging client-server communications. It would be helpful if you could use your home computer as a web server.

But there is a snag. A commercial Internet Service provides a *static* Internet Protocol address. When a user types an address into a browser, that address is sent to specialized servers on the Internet, Domain Name Servers. They locate the static IP address corresponding to the domain name.

If you are connected to the Internet via a cable or DSL service in your home, you don't have a static IP address. Instead, you have an address that can change every time you start your computer. It may even change while your computer is running. You can't have a domain name assigned to your home server.

This article shows a way that you can use a commercial Internet Service Provider account to act as your personal name

server. The user enters the address of a web page on the commercial account. That page redirects the request to your home computer.

The load on the commercial server is very low. Once it's set up, you never need to change it. You may be able to use a friend's account. The only requirement for the commercial server is that it can run scripts in perl or some similar language. All the changes that you make on your site after that are on the home Macintosh.

The commercial server needs to know the Internet Protocol address currently assigned to your home computer. A script running periodically on home computer periodically sends a message to the commercial server to inform it of the home computer's address.

The method has its drawbacks. I'm not suggesting this as a replacement for a static IP address for commercial or high volume applications. It is, however, handy for demonstrations and uses that don't demand 100% reliability.

Overview

The basic idea is very simple. This process uses two web servers: one on your Macintosh at home (hereafter called the *Home Server*), and a commercial server (hereafter called the *Redirection Server*). The Home Server periodically runs the script

ip_to_server.pl. That script sends a request to the Redirection Server. The request contains the Home Server's IP address. The Redirection Server responds with the script ip_to_file.pl which saves the Home Server's IP address in a file. When a browser sends a request to the Redirection Server, the Redirection Server uses the saved IP address to redirect the client's request for a web page to the Home Server. (See Figure 1.)

For example suppose that the saved IP address is 92.2.92.2 and the address of the Redirection Server is http://stowlake.com/. The user enters or clicks on a link to: http://stowlake.com/redirect.pl?xfile=some.html. The script redirect.pl redirects the request to the file specified by the xfile parameter on the Home Server: http://92.2.92.2/some.html

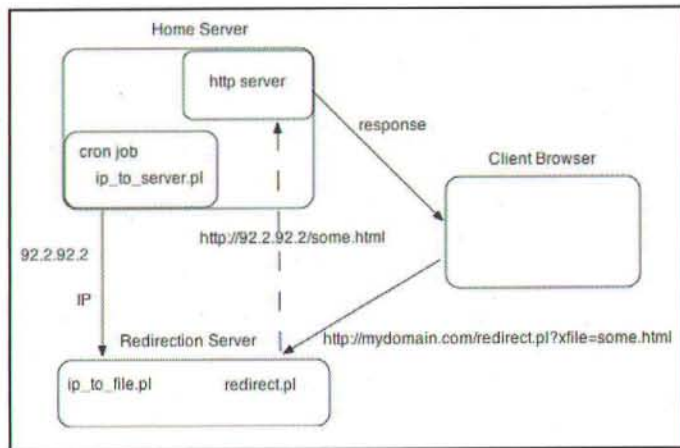


Figure 1. Putting it all together.

This method has several limitations. First, of course, the Home Server needs to be turned on all the time that someone may want to access the pages involved. Also, there can be a lag between the change in the IP address on the Home Server and latest cycle of saving the IP address on the Redirection Server. When that happens, the client will fail to get the right page. The frequency with which a dynamic IP address changes depends upon the quality of the DSL connection. It can vary between several times per hour to several times a week. Do some experiments.

Configuring the Redirection Server

Let's get into specific details. First, let's see how to configure the Redirection Server. You can use any commercial Internet server that supports Perl or some other convenient language. Our examples are in Perl. One script stores the IP address of the Home Server. Another retrieves that address when it needs to redirect a request from the web client.

Remembering the Home Server's IP Address

In response to a request from the Home Server, the script ip_to_file.pl stores the address of the sender. Listing 1 shows this script.

Listing 1: Caching the Home Server's Address ip_to_file.pl

```
#!/usr/bin/perl -w
# Save IP address received from Home Server to file.

use CGI qw(:standard);

$ipfile = 'home_server_ip';
print header, start_html;
if (open IPFILE, ">$ipfile")
{
    $ip = $ENV{REMOTE_ADDR};
    print IPFILE $ip . "\n";
    close IPFILE;
    print "Wrote to file $ipfile: $ip";
}
else
{
    print "Failed to open file $ipfile for writing.";
}
print end_html;
```

This script is a CGI script that sends back a response that contains the IP address that it just saved.

Redirecting to the Home Server IP Address

The redirection script redirect.pl is shown in Listing 2. It takes a single parameter, 'xfile'. That is the filepath on the Home Server to which the request will be redirected.

Listing 2: Redirecting to the Home Server's address redirect.pl

```
#!/usr/bin/perl -w
# Save IP received from Home Server.

use CGI qw(:standard);

# Redirect a request to the Home Server.
# The file to which to redirect is specified by parameter xfile.
# The IP address of the Home Server is retrieved from the file.

$ipfile = 'home_server_ip';
$fileOnHomeServer = param('xfile');

if (defined($fileOnHomeServer))
{
    if (open IPFILE, "<$ipfile")
    {
        chomp ($ip = <IPFILE>);
        print "Location: http://$ip/$fileOnHomeServer\n\n";
    }
    else
    {
        print header, start_html;
        print "Failed to open file $ipfile";
    }
}
else
{
    print header, start_html;
    print "Failed to retrieve file to access.";
    $fail=1;
}
print end_html;
```


Configuring the Home Server

The script in Listing 3, `ip_to_server.pl`, sends a request to `ip_to_file.pl` on the Redirection Server. The IP address of the sender is automatically sent with an HTTP request. There doesn't need to be any special code in `ip_to_server.pl` for that.

Listing 3: Sending an address to the Redirection Server `ip_to_server.pl`

```
#!/usr/bin/perl -w

use LWP::UserAgent;
use HTTP::Request::Common qw(GET);

# Send a request to the Redirection Server to save this computer's IP
#
# Server finds the IP in the request, saves it and returns
# the IP in its response.
#
# This script records its transactions in a log file.
#
$RedirectionServerAddress =
'http://www.stowlake.com/cgi-bin/ip_to_file.pl';
open LOGFILE, ">>/tmp/ip_to_server.log"
or die "Couldn't open ip_to_server.log $!\n";
print LOGFILE scalar(localtime());
my $userAgent = new LWP::UserAgent;
my $request = GET $RedirectionServerAddress;
my $response = $userAgent->request($request);
my $success = $response->is_success();
my $found = 0;
if( $success )
{
    my $content = $response->content;
    if ( $content )
    {
        my @data = split /\n/, $response->content;
        for(@data)
        {
            if(/Wrote to file/)
            {
                print LOGFILE "    ip was updated to: ";
                /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/;
                print LOGFILE $1."\n";
                $found = 1;
            }
        }
    }
}
else
{
    print LOGFILE "Failed to contact server: \n",
        $response->error_as_HTML."\n";
}
print LOGFILE "failed to update ip\n" if !$found;
```

Setting It Up

So we've got a couple of Perl scripts at hand. How do we set this thing in motion?

Upload Scripts to the Redirection Server

First, we need to upload the two scripts `ip_to_file.pl` and `redirect.pl` to the redirection server. Most ISP's allow you to upload files using the File Transfer Protocol (FTP). I typically use the application Fetch 4.0.3 to do this. You may of course prefer other tools.

Depending on your ISP, there may be a specific folder into which you put CGI scripts. For mine, `cgi-bin` is its name and it appears when I connect to their FTP site. Check with yours. They may set up the files differently. Whatever folder you use, you need to set the file permissions to allow execution of the scripts. In Fetch, use "Set Permissions..." in the Remote menu. Check the boxes as shown in Figure 2 for each of the Perl scripts.

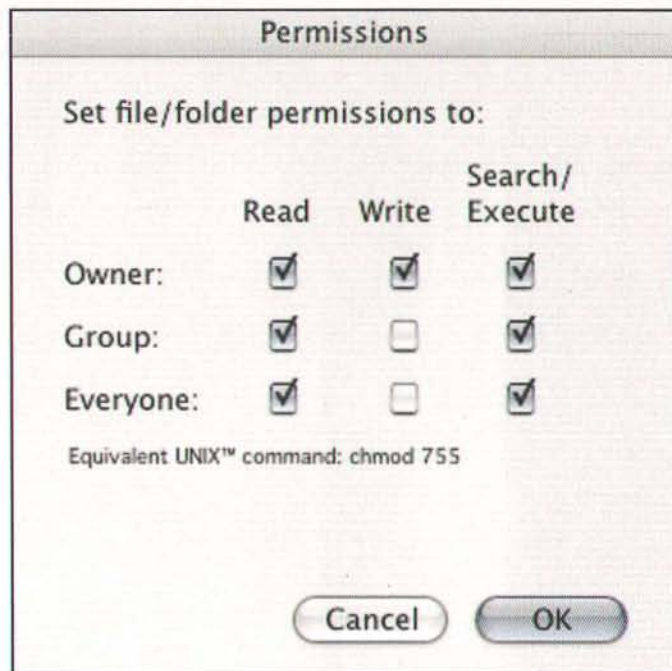


Figure 2. Setting File Permissions

If you have access to the UNIX shell on your Redirection Server, you can set permissions with the shell commands:

```
chmod 755 ip_to_file.pl
chmod 755 redirect.pl
```

Once the files are in place, you can test them from a web browser. Enter the URL for `ip_to_file`. In my case, it's:

http://www.stowlake.com/ip_to_file.pl

A message with the IP address of your home Macintosh should appear in the browser window. Look for a file on your ISP site called `home_server_ip`. It should be in the same directory as `ip_to_file.pl`.

Start the Web Server on the Home Macintosh

Next we need to start the server on the home machine. Open the System Preferences application and choose the Sharing panel. Check the box labeled Personal Web Sharing, as in Figure 3.

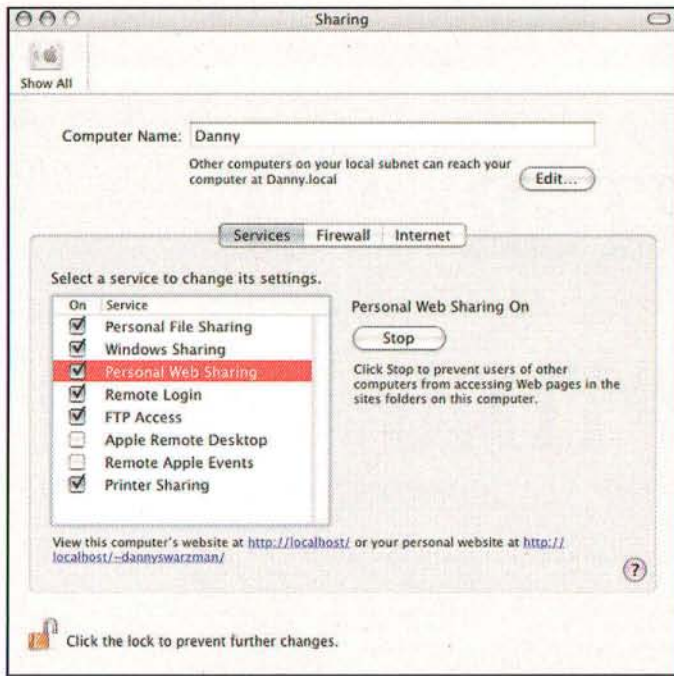


Figure 3. Starting the Web Server

Put a sample web page into the directory `/Library/WebServer/Documents` on the boot drive. If your file is called `some.html`, type this into a web browser's address box: `http://localhost/some.html`

If you see the page, your web server is running.

Run a Script on the Home Macintosh Periodically

The final piece of the puzzle is to run the `ip_to_server.pl` script on the Home Server. Modify the sample file to use your Redirection Server by changing the value of `$kRedirectionServerAddress` to the address of the directory on the Redirection Server that contains the scripts you put there. Put `ip_to_server.pl` in a convenient place on your Macintosh. I chose `/Library/WebServer/CGI-Executables`.

Make the script executable. Open the Terminal application. The Terminal application is in the Utilities subfolder of the Applications folder in the main drive. Go to the directory to where you put `ip_to_server.pl` and change the permissions. For example:

```
cd /Library/WebServer/CGI-Executables
chmod 755 ip_to_server.pl
```

Now try running the script. Just enter:

```
./ip_to_server.pl
```

You can tell if it worked with this command:

```
cat /tmp/ip_to_server.log
```

You should see a message in the terminal window telling you that the IP address was updated.

Finally, you need to get the script to run periodically, in my case every 10 minutes. You need to use the command, `crontab`, in

the Terminal application. There is a program that is always running, the `cron` daemon. It periodically scans the table created by the `crontab` command. For each entry, if its time has come, `cron` calls the associated command. (You may want to review `crontab` documentation. Type "`man crontab`" into the Terminal application. Or, you can use the application `ManOpen` to view man pages.)

The first step is to create a file that specifies when `ip_to_server.pl` is to run. Let's call it `ip_to_server.crontable`. This example says: execute the specified command at 0 minutes, 10 minutes, ..., after the hour. Repeat every hour of every day of every month, including every day of the week. The contents of `ip_to_server.crontable` is this one line:

```
0,10,20,30,40,50 * * * * /Library/WebServer/CGI-
Executables/ip_to_server.pl
```

Next enter a command in the terminal window:

```
sudo crontab -u username ip_to_server.crontable
```

(For `username`, substitute your login name on your Macintosh.)

Finally to verify:

```
crontab -l
```

The last command displays contents of the `cron` table just created.

Conclusion

Many of us may have thought that signing up for Internet services that use a dynamic IP address was thereby to give up any hope of serving web pages from our home computers. As I've shown in this article, a dynamic IP address is really no barrier to doing this. Rather, we simply need to upload a few simple Perl scripts to the Redirection Server and then periodically update the stored IP address by running a local `cron` job. As we've seen, this is not an industrial-strength solution, but it's probably fine for letting your relatives view photos from your recent trip to Hawaii or peruse your weblog. You need to experiment to know how it works for you.

References and Credits

`ManOpen` is a Macintosh application to easily view man pages. You can scroll through them; follow links from man page to man page and print man pages.

<http://www.clindberg.org/projects/ManOpen.html>

Bob Ackerman showed me how to do what I've described here. The sample code in this article was inspired by his example. He also reviewed this article. Thanks also to Victoria Leonard for her helpful suggestions and to Tim Monroe for his editing assistance.



About The Author

Danny Swarzman writes programs in JavaScript, Java, C++ and other languages. He also plays Go and grows vegetables. You can contact him with comments and job offers at <mailto:dannys@stowlake.com> and <http://www.stowlake.com>.

USER INTERFACE SCRIPTING

As we have seen in the past, AppleScript is a great tool for creating some pretty amazing automated workflows. When implementing an AppleScript-based workflow, you are really only limited by your imagination, and by the AppleScript support that is available in the applications that you want to automate.

Many scriptable applications offer enough AppleScript support for the types of tasks that you would want to automate the most. However, at times, you may find yourself needing to script an application with limited AppleScript support, and that one task you *really* need to automate just is not accessible through scripting. Or, worse yet, the application you want to automate is not scriptable at all!

How do you handle these limitations? Do you simply give up? I think not. For starters, you might check around to see if there are other scriptable applications that can be substituted to automate the same task.

Another option is to consider trying to automate the application by writing AppleScript code that interacts directly with the application's interface itself. Fortunately, with the release of Mac OS X 10.3, Apple introduced a new AppleScript feature, user interface scripting, or *UI Scripting*, which can be used to do just that.

What is UI Scripting?

UI Scripting is a feature in Mac OS X that provides a way for AppleScripts to interact directly with interface elements in

running applications. For example, you might write a script that clicks a button in a window, selects a menu item in a popup menu, or types text into a text field of a non-scriptable application.

The AppleScript terminology for *UI Scripting* is not built into AppleScript itself, nor is it included in a scripting addition. Rather, *UI Scripting* is part of *System Events*, a background application that provides a central location for interacting with various aspects of Mac OS X itself, including running processes, or applications.

In order to view the AppleScript terminology for *UI Scripting*, open the dictionary for *System Events* by double clicking on it in the *Script Editor's Library* palette.

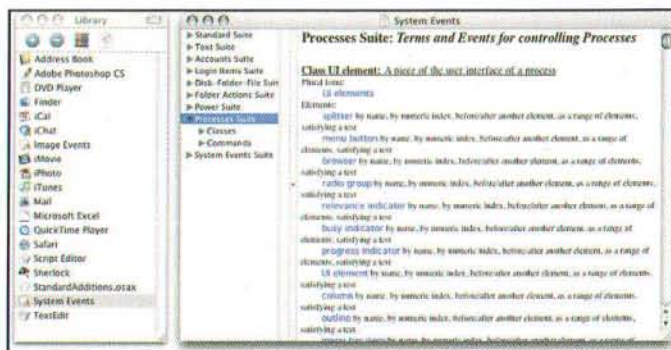


Figure 1. The Script Editor Library Palette and the System Events Dictionary

If you are using a different script editor, or prefer to open the dictionary by selecting *File > Open Dictionary* in the *Script Editor* menu bar, *System Events* can be found in the *System >*

Library > CoreServices folder on your machine. In the *System Events* dictionary, *UI Scripting*-related terminology can be found under the *Processes Suite*. We will explore the *UI Scripting* terminology found in this suite shortly.

Enabling UI Scripting

Before writing code that makes use of *UI Scripting* in Mac OS X, you will first need to manually configure the machine on which the script will run, so that it can respond to *UI Scripting* commands. To enable *UI Scripting*, launch the *System Preferences* application, and click on the *Universal Access* button. Next, enable the checkbox labeled *Enable access for assistive devices*. See figure 2.

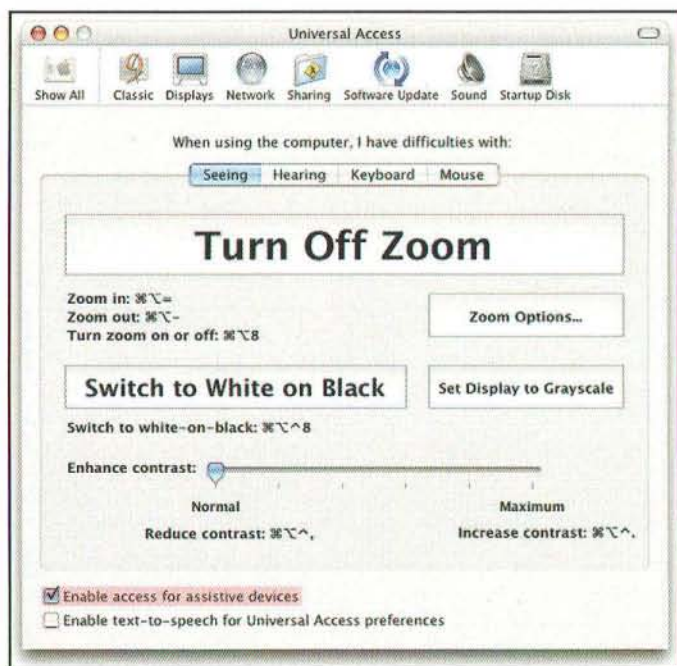


Figure 2. Universal Access Preference Pane

If the final destination for your script is another user's machine, then you should make sure that you provide that user with instructions for enabling *UI Scripting* on the machine prior to running the script. In order to ensure that the user actually follows your instructions, you may also want to add some code into your script to verify that *UI Scripting* has been enabled, and prevent the script from proceeding if it has not been enabled. You can tell if *UI Scripting* has been enabled by checking the value of the *UI elements enabled* property of the *System Events* application. This property will contain a *true* or *false* value, indicating whether or not *UI Scripting* is enabled.

The following example code will check to see if *UI Scripting* is enabled. If it is not enabled, the script will launch

System Preferences, display the *Universal Access* pane, and notify the user that the *Enable access for assistive devices* checkbox must be enabled in order to proceed.

```
tell application "System Events" to set
isUIScriptingEnabled to UI elements enabled
if isUIScriptingEnabled = false then
    tell application "System Preferences"
        activate
        set current pane to pane
        "com.apple.preference.universalaccess"
        display dialog "Your system is not properly
        configured to run this script. Please select the
        \"Enable access for assistive devices\" checkbox and
        trigger the script again to proceed."
    return
    end tell
end if
```

— Add your code here, which will be triggered if *UI Scripting* is enabled

UI Scripting Terminology

As previously mentioned, the AppleScript terminology for *UI Scripting* is found under the *Processes Suite* in the *System Events* dictionary. Here, you will find a variety of classes, or objects, along with a few commands. Let's briefly discuss the commands first. We'll see them in action a little later.

Commands

There are only a handful of commands that you will use for *UI Scripting*, and they are *click*, *key code*, *keystroke*, *perform*, and *select*. In this article, I am going to touch on the *click* and *keystroke* commands, as they are what I have found to be particularly useful, and I have chosen to include them in the examples at the end of this article.

As one would expect, the *click* command is used to simulate the user clicking on an interface element. For example, you might use the *click* command to click a button on a window.

```
click button 1 of window 1
```

The *keystroke* command is used to enter keystrokes, as if they were typed by a user. The *keystroke* command also offers the ability, through an optional parameter, to specify modifier keys that should be invoked when the command is executed. For example, you may need to simulate the process of holding down the *command* key while typing a key.

```
keystroke "P" using {command down}
```

Again, these are just very brief introductions to two commands that can be used to interact with interfaces. You should spend some time exploring the other commands on your own, in order to become familiar with

them. Now, let's take a look at some of the classes listed under the *Processes* suite of the *System Events* dictionary.

The Process Class

When working with *UI Scripting*, the top-level class you will address is the *process* class. A *process* signifies any process, or application, that is currently running on the machine. To get the names of any currently running processes, use the following syntax:

```
tell application "System Events" to return name of every process

-> ["loginwindow", "Dock", ... "Finder", "Preview", "Script Editor"]
```

Processes possess a variety of properties, which can be accessed through AppleScript. One such property is the *frontmost* property. In some cases, you may need to bring a process to the front in order to interact with its interface. This can be done by setting the *frontmost* property of the process to a value of *true*. For example, the following code brings the application *Preview* to the front.

```
tell application "System Events" to set frontmost of process "Preview" to true
```

Note in this example that I refer to the *Preview* application as a *process* of the *System Events* application. Be sure to explore the other properties of processes, as well, as they can provide other important information to your script.

I have chosen to use *Preview* for the examples in this article, since *Preview* is not AppleScriptable, and since it is installed with Mac OS X.

UI Element Classes

A process with an interface typically contains a variety of interface elements, such as windows, buttons, text fields, and more. These are the elements with which you will primarily want your script to interact.

In the *System Events* dictionary, a *UI Element* is, in itself, a class, and possesses a variety of properties common to most interface elements. For example, since many interface elements possess the ability to be enabled or disabled, an *enabled* property can be found under the more generic class of *UI Element*. *Processes* also inherit some properties from this class.

There are a number of different classes of interface elements in the *Processes* suite. If you have used AppleScript Studio in the past, then you will no doubt be familiar with the names of at least a few of them, although the exact naming may be slightly different from that of AppleScript Studio. Some of the more commonly accessed classes of interface elements include checkboxes, menus, menu items, pop up buttons, radio buttons, text fields, and windows. As mentioned above, since all interface elements possess certain characteristics, each class of interface element inherits properties from the *UI Element* class.

Be sure to browse through the various classes of interface elements listed in the *Processes* suite.

Referring to UI Elements

Just like scripting any object in a scriptable application, when writing code to interact with an interface, you must address the interface elements within their object hierarchy. Deciphering an object's hierarchy within a process' interface is usually no small feat. For basic interfaces, it may be as simple as referring to an interface element on a specific window. However, for more complex interfaces, it can quickly become a long set of nested object references.

Let's take a look at an example. The following code will click the *Drawer* button in the tool bar of an opened document in *Preview*, toggling the drawer opened or closed.

```
tell application "System Events"
  tell process "Preview"
    click button "Drawer" of tool bar 1 of window 1
  end tell
end tell
```

You can see from this example that in order to interact with the *Drawer* button, you must indicate specifically where the button is located within the interface. In this case, it is on the tool bar of the front window.

There is some good news, though. Apple doesn't leave you high and dry to figure out these object hierarchies on your own. There is a tool available to help. It is called *UI Element Inspector*, and it can be downloaded from Apple's AppleScript web site <<http://www.apple.com/applescript/uisc scripting/>>.

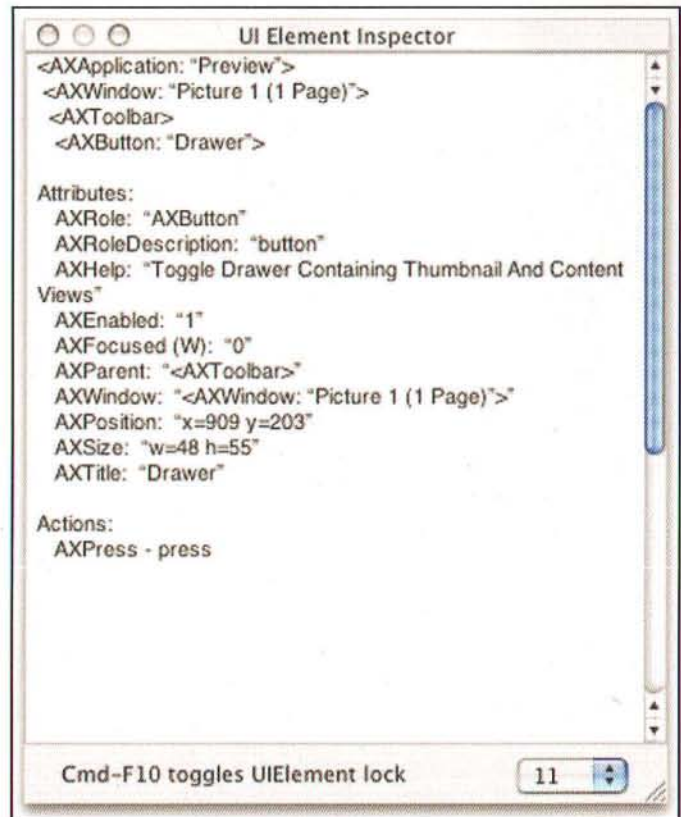


Figure 3. UI Element Inspector

When launched, *UI Element Inspector* floats above all interfaces, and displays information about the interface elements located directly below the mouse pointer. Information displayed includes the location of the interface element, within its object hierarchy, along with values from various properties. In figure 2, *UI Element Inspector* is indicating the exact location of the *Drawer* button in a *Preview* window – within a tool bar, which is within a window of the application. As you can see, the tool bar itself does not possess a specific name, which is why I referred to it as *tool bar 1* in my code.

Examples

Now, let's take a look at some examples of *UI Scripting* in action. The following example code assumes that you have a document opened in *Preview*. When this code is run, it will bring *Preview* to the front and print the front window.

```
tell application "System Events"
  tell process "Preview"
    set frontmost to true
    click menu item "Print..." of menu "File" of menu bar 1
    repeat until sheet "Print" of window 1 exists
      end repeat
    keystroke return
  end tell
end tell
```

The following example code assumes that you have a document opened in *Preview*. When this code is run, it will bring *Preview* to the front and export the front window to the desktop in TIFF format.

```
tell application "System Events"
  tell process "Preview"
    set frontmost to true
    keystroke "E" using {command down, shift down}
    repeat until sheet 1 of window 1 exists
      end repeat
    tell sheet 1 of window 1
      keystroke "Output"
      keystroke "D" using command down
      tell pop up button 1 of group 1
        click
        tell menu 1
          click menu item "TIFF"
        end tell
      end tell
      delay 1
      click button "Save"
    end tell
  end tell
end tell
```

Limitations

UI Scripting is based on Mac OS X's accessibility framework, which is why it must be enabled from the *Universal Access* preference pane. Because it is based on the accessibility framework, *UI Scripting* can only interact with applications that have been updated to support this framework. The accessibility framework was first released with Mac OS X 10.2, so many applications have accessibility support built-in by now. However, some applications do not, and may not respond to *UI Scripting* terminology. To automate applications that do not support *UI Scripting*, you may need to turn to other tools, such as *QuickKeys* <<http://www.quickkeys.com>>.

A Third Party Assistant

While *UI Element Inspector* provides some much needed input for writing your AppleScript code, a third-party application takes it a step further.

PreFab UI Browser, a commercial application available from PreFab Software, Inc., offers all of the functionality of the *UI Element Inspector*, and a lot more. *PreFab UI Browser* provides robust interfaces designed for the serious scripter, making it easy to access information about the complete interface of any accessible running application. Other features of *PreFab UI Browser* include the ability to trigger hot keys to enable and disable interface element browsing, auto-generation of AppleScript code to interact with interface elements, and much more. Be sure to visit the PreFab Software, Inc. web site <<http://www.prefab.com>> for more information about this tool, or to download a demo version. While you're there, be sure to check out another product – *PreFab UI Actions*. This tool will allow you to configure AppleScripts to trigger automatically whenever specific user interface actions occur in an application. For example, you could configure an application to trigger a script whenever a user selects the *Save As* from the *File* menu. The possibilities are limitless.

In Closing

As you can see, *UI Scripting* can be quite a useful tool when trying to automate an application that doesn't provide the AppleScript support that you need. So, before you decide that just because an application isn't scriptable, it can't be automated with AppleScript, be sure to check out its interaction with *UI Scripting*.

In my experience, there is not much on the Mac that can't be automated... one way or another. Until next time, keep scripting!

MM

About The Author



Benjamin Waldie is president of *Automated Workflows, LLC*, a firm specializing in AppleScript and workflow automation consulting. In addition to his role as a consultant, Benjamin is an author for *SpiderWorks, LLC* and an evangelist of AppleScript. He can often be seen presenting at Macintosh User Groups, Macworld Expos, and more. For additional information about Benjamin, please visit <http://www.automatedworkflows.com>, or email Benjamin at applescriptguru@mac.com.



Introduction

This is the first part of a (two-part) article showing how to use a few of the great new pieces of Cocoa functionality available to developers with the release of Tiger. The first—the focus of this article—is NSXML, a set of Cocoa classes that implement a robust tree-based XML parser that provides the ability to create and edit XML from scratch as well as from data contained in files and vended through web services.

The second new technology, which we'll concentrate on in the next installment, is Core Data, one of the most astoundingly wonderful bits of technology Apple has ever given to the Mac development community.

In this month's article, we'll build a small Cocoa application that will retrieve book information from Amazon's XML-based web service, parse the vended data into Cocoa objects, then let the user edit the data before saving it to an XML file on their local disk.

Next month, we'll use Core Data and Cocoa Bindings to make a version of the same application with more functionality that's easier to maintain and support, takes less time to build, and requires writing much less code.

It's important to note here that you must have Tiger *and* the Tiger development tools installed, since neither NSXML nor Core Data are available in earlier releases. So, fire up Xcode 2.0, create a new Cocoa Application project titled **Amazon-XML** and let's dive in.

The Data model

Our first step is to build our data model the traditional Cocoa way. We have to create Cocoa classes to hold our data. We'll also need to write code that will allow instances of our data model objects to be copied, compared, and persisted. Go ahead and create a new objective-C class called **MTBook** in your newly created project. Now give **MTBook** instance variables to hold our book data.

MTBook Instance Variable Declaration

We're going to store a subset of the data that Amazon's web service provides. The bibliography provides a reference to the Amazon SDK where you can find out more about their SDK and XML format. The last two instance variables do not come from Amazon, but rather are there to hold additional, user-provided information.

```
@interface MTBook : NSObject
{
    NSString      *isbn;
    NSString      *title;
    NSString      *publisher;
    NSDate        *dateReleased;
    NSData        *coverImage;
    NSMutableArray *authors;
    int           salesRank;
    NSURL         *url;
    NSDate        *dateRead;
    NSString      *comment;
}
```

Of course, we need to add accessor and mutator methods for each of these variables, as well as code to initialize the

object, copy it, persist it, restore it, compare it, and release its memory when it's deallocated. These aspects of building a data model are some of the very few tedious tasks in programming Cocoa. Next month you'll see that these tedious steps are no longer necessary thanks to Core Data. But let's not get ahead of ourselves.

These various methods are not shown in code listings; they are standard Cocoa tasks that have been written about extensively elsewhere. I used Kevin Callahan's handy Accessorizer program to automatically generate the accessors and mutators, then hand-coded the initializers as well as the other methods a data object needs, such as code to copy an instance, compare one instance to another, or to encode and decode an instance. The process of creating the **MTBook** class took a considerable amount of time, even though it's all very straightforward. If you don't feel like doing all those tedious tasks (and who would blame you?), feel free to copy **MTBook** from the issue CD.

You may be wondering why we would choose to store the cover image as an **NSData** instead of an **UIImage**. The simple reason is that **UIImage** will not necessarily preserve the original image data at the same size and resolution. To avoid recompression and possible loss of image quality or resolution, we'll store the original JPEG data as retrieved.

Architecture of NSXML

NSXML's functionality is provided by a small collection of classes, only a couple of which you will need to use.

Most developers will only ever need to interact with **NSXMLNode**, **NSXMLElement**, and **NSXMLDocument**, though some may use **NSXMLDTD**, and **NSXMLDTDNode** in their travels. Since our application is not concerned with creating DTDs and we are going to assume that Amazon's XML validates correctly, we only need to look at the first three:

NSXMLDocument represents an entire XML document as a Cocoa object;

NSXMLElement represent the various, nested nodes of an **NSXMLDocument**

NSXMLNode represent XML "nodes". Nodes can be just about any item in an XML document (including the document itself), and as a result, **NSXMLNode** is the parent class of both **NSXMLDocument** and **NSXMLElement**. Most methods you'll interact with will return an **NSXMLNode**.

Nodes can contain attributes and namespaces. An attribute is data stored within the node's opening XML tag using key-value coding and is generally used to store metadata about the node. A namespace is a special kind of attribute used to avoid name conflicts among elements and attributes. Most XML you'll encounter probably won't use namespaces or will use a single, standard namespace. As a practical matter, you can often ignore namespaces when parsing XML. Amazon's XML uses a single, standard W3C namespace, so we're going to ignore them from here on in.

In addition to attributes and namespaces, **NSXMLNodes** can contain child nodes. Its children can, themselves, contain child nodes (and so on). This nesting behavior gives XML a lot of its flexibility and power and is what makes a tree-based parser such a good choice for many situations.

We need to take note of two other boxes on this diagram above. One is the light green box on the far right that is titled (**utils**). We'll talk about that one shortly. The other is on the left side of the diagram and is titled **NSXMLParser**. The downside of a tree-based XML parser is that it uses a fair chunk of memory and is overkill for many situations (including ours, if it were a real application). **NSXMLParser** is a standalone class that implements an event-driven XML parser. It's easy to use and has a much smaller footprint than **NSXML**. **NSXMLParser** is beyond the scope of this article, but you should know it's there and use it when appropriate.

Handy Additions to NSXML

There is one thing that NSXML lacks that many other tree-based XML parsers offer: the ability to retrieve a child of a given node by its name. Since referring to children by name makes code much more readable, I've created a category on **NSXMLNode** that adds this functionality to **NSXMLNode** objects.

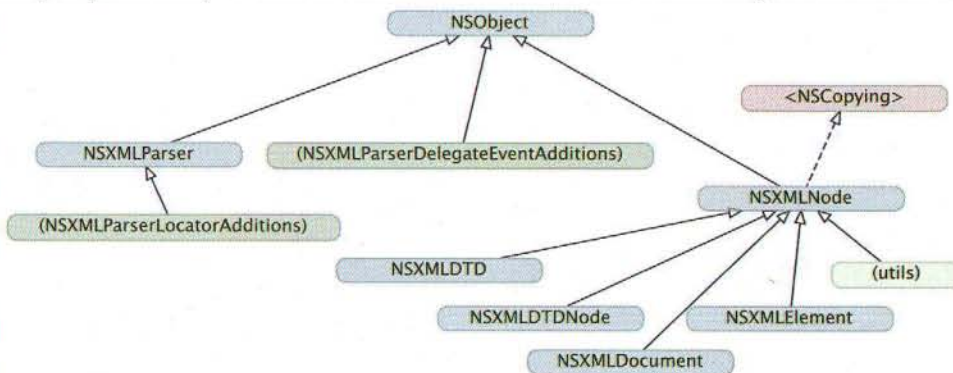


Figure 1. NSXML Class Diagram

Listing 1: NSXMLNode-utils.m

NSXMLNode utilities

This category on **NSXMLNode** adds two bits of useful functionality to **NSXMLNode**. The first method allows retrieving a child node of a given name. The second converts all the children of a given node to their string values, allowing you to retrieve an array of a node's children with one call. You'll see how both of these are used later in the article.

```
@implementation NSXMLNode(utils)
```

```
- (NSXMLNode *)childNamed:(NSString *)name
{
    NSEnumerator *e = [[self children] objectEnumerator];
```



```

NSXMLNode *node;
while (node = [e nextObject])
    if ([node name] isEqualToString:name))
        return node;

return nil;

- (NSArray *)childrenAsStrings
{
    NSMutableArray *ret = [[NSMutableArray arrayWithCapacity:
        [[self children] count]] retain];
    NSEnumerator *e = [[self children] objectEnumerator];
    NSXMLNode *node;
    while (node = [e nextObject])
        [ret addObject:[node stringValue]];

    return [ret autorelease];
}
@end

```

Building the Interface

The next step in building our application is to design the interface in Interface Builder. If you're comfortable with the basics of using Interface Builder and how to link them to outlets and actions in your code, feel free to skip ahead to the next section: Coding with NSXML.

We're going to build a relatively simple interface here, just enough to work with. First, we'll lay out our interface so that it has fields corresponding to each instance variable in our data model, along with a button to initiate the search, and another to save the revised XML to a file. Mine looks like this:

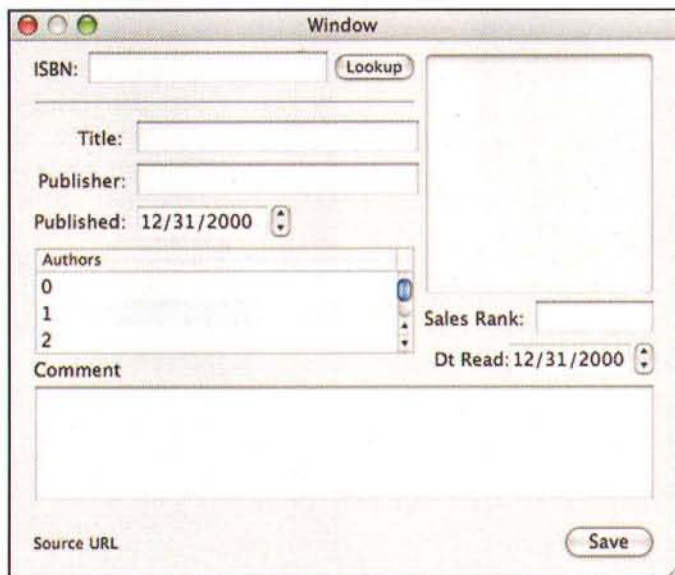


Figure 2. Our Window in Interface Builder

Next, we'll go to the Classes tab, right-click `NSObject` and create a subclass of `NSObject` which we'll call `MTXMLAppDelegate`. This new class will act as both our application delegate and as our main window's controller class. Control-drag from the `File's Owner` icon to the instance of `MTXMLAppDelegate` and bind it to the delegate outlet. While we're here, control-drag from the `Authors` table to `MTXMLAppDelegate` and link to the `DataSource` outlet.

Double-click on the `MTXMLAppDelegate` instance icon, which will open the object inspector where you can add outlets and actions. Add actions called `doSearch:`, and `saveToXML:`. Also add the outlets shown in Figure 3, which correspond to each of the user interface elements that we'll need access to in our code.

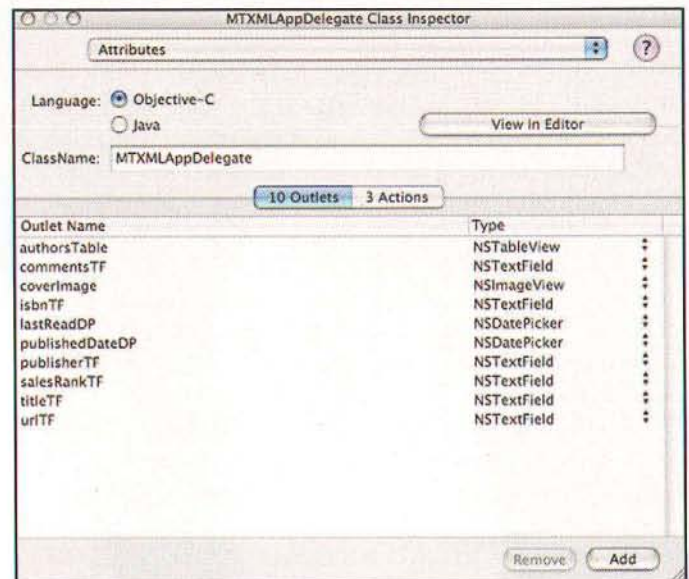


Figure 3. Outlets

Now control-drag from the `Lookup` button to the `doSearch:` outlet of `MTXMLAppDelegate` and repeat with the `ISBN` text field. Now control-drag from the `Save` button and bind to the `saveToXML:` outlet. If we were creating a real application where it was important to keep the data model synchronized with the interface in real-time, we would also have to have an additional method that was called when any of the other field values changed. I mention this only because real-time synchronization of the data model is yet another thing you get for free using Core Data. For our limited purposes, we'll simply update the model when the `Save` button is pressed.

We also need to link our application delegate's outlets to the various fields in our window, so control-drag from the `MTXMLAppDelegate` instance icon to each of the fields and bind to the corresponding outlet. Once that is done, we're ready to write some code. Go ahead and generate the source files for `MTXMLAppDelegate`, then go back to Xcode, open up `MTXMLAppDelegate.h`, and immediately following all the automatically generated `IBOutlet` variables generated, add an `MTBook` instance variable called `book`, and don't forget to include `MTBook.h`. Our `doSearch:` method is where we'll actually retrieve the XML data from Amazon and work with it. A stub implementation of this method has been generated for us, so we'll add our code there.

Coding with NSXML

As with most Cocoa classes, the `NSXML` classes have been designed to be easy to use and to insulate you from having to

know too much about the inner workings of the class. To instantiate an `NSXMLDocument` based on data retrieved from a URL, you simply allocate as normal and then initialize the object using `initWithContentsOfURL:options:error:`. If you already have the XML in memory, you can initialize instead with `initWithData:options:error:` instead.

The root node is available from an `NSXMLDocument` by using the `rootElement` method. An array of the child nodes of any element can be retrieved using the `children` method of `NSXMLNode`, or you can request a specific child by name using the `childNamed:` method from our `utils` category. To get the string contents of a node – the value between the begin and end tags—simply call the `stringValue` method.

The easiest way to show how these various methods work is to show some code.

Listing 2: MTXMLAppDelegate.m

doSearch:

This method uses `NSXML` to parse the results of the XML data retrieved from an Amazon ISBN search then stores that data as an `MTBook` object.

```
(IBAction)doSearch:(id)sender
{
    // More information about Amazon's XML web services is available from
    // http://tinyurl.com/cb89g

    // Here, we're just setting up the Amazon URL based on our assigned
    // developer token
    // (assigned to us by Amazon) and the ISBN number entered by the user

    NSString *urlBase =
        @"http://xml.amazon.com/onca/xml3?t=l&dev-
        t=%&AsinSearch=%&type=heavy&f=xml";
    NSString *urlString = [NSString stringWithFormat:urlBase,
        AMAZON_DEV_TOKEN,
        [isbnTF stringValue]];
    NSURL *theURL = [NSURL URLWithString:urlString];

    // NSXML doesn't throw an exception on bad allocation, nor does it
    // simply return a
    // nil. NSXML acts old-school in some ways; it wants a pointer to a variable
    // which it
    // will populate with an error code if there's a problem

    NSError *err=nil;

    // Initialize our document with the XML data in our URL

    NSXMLDocument *xmlDoc = [[NSXMLDocument alloc]
        initWithContentsOfURL:theURL
        options:nil
        error:&err];

    // If we were doing proper error checking, we'd check the value of err
    // here. We'll
    // skip it and simply check for xmlDoc == nil later. This method works
    // fine if you
    // don't intend to give the specifics of any problems encountered to the
    // end user.

    // Get a reference to the root node

    NSXMLNode *rootNode = [xmlDoc rootElement];

    // In case of an error, Amazon includes a node called ErrorMsg, its
    // presence tells us
```

```
// that an error happened not in parsing the XML, but rather on Amazon's
// side, so we
// check for it

NSXMLNode *errorNode = [rootNode childNamed:@"ErrorMsg"];

if (rootNode == nil || errorNode != nil)
{
    // We'll provide an error message in the URL field, blank the others, then
    // abort.
    // objectValue is a method that will give you the contents of a node as
    // an
    // NSObject, which will always be a string when first parsed from
    // XML,
    // but which might be another type of Cocoa object such as an
    // NSDate
    // if you created the NSXMLDocument programmatically as we'll do
    // later.
    // NSXMLDocument is smart enough to automatically convert
    // commonly used
    // objects to the correct W3C formatting when outputting them to
    // XML.
    [urlTF setStringValue:[errorNode stringValue]];
    [commentsTF setStringValue:@""];
    [coverImage setImage:nil];
    [isbnTF setStringValue:@""];
    [publisherTF setStringValue:@""];
    [salesRankTF setStringValue:@""];
    [titleTF setStringValue:@""];
    return;
}
else
{
    // If no error, the book information is contained in a node called
    // Details, so we'll
    // grab it. Though the method actually returns an NSXMLNode, we
    // know that, as a
    // child of a document, we'll actually get an NSXMLElement. We
    // declare it as an
    // NSXMLElement to avoid casting it later

    NSXMLElement *detailsNode = [rootNode
        childNamed:@"Details"];

    // Release memory from any prior book, and start with a fresh object
    [book release];
    book = [[MTBook alloc] init];

    // Now set the fields of our data model from singleton child nodes
    // NSStrings are easy - we just call the stringValue method on the node

    [book setIsbn:
        [[detailsNode childNamed:@"Asin"] stringValue]];
    [book setTitle:
        [[detailsNode childNamed:@"ProductName"]
        stringValue]];
    [book setPublisher:
        [[detailsNode childNamed:@"Manufacturer"]
        stringValue]];

    // Since we created this NSXMLDocument from parsed XML, we're
    // going to get
    // NSStrings when we call objectValue, so dates, numbers, and others
    // have
    // to be manually converted from strings

    [book setDateReleased:[NSDate
        dateWithNaturalLanguageString:[detailsNode
        childNamed:@"ReleaseDate"] stringValue]];

    // Sales rank is a scalar value
```



```

[book setSalesRank:[[[[detailsNode
    childNamed:@"SalesRank"] stringValue] intValue]];

// URL is stored as an attribute of the Details node, not as child
element.The
// syntax for getting attributes is slightly different than for child
elements

[book setUrl:[NSURL URLWithString:[[[detailsNode
    attributeForName:@"url"] stringValue]]];

//The XML we retrieved doesn't have the cover image, but it does have
a URL.

NSURL *imageUrl = [NSURL URLWithString:[[[detailsNode
    childNamed:@"ImageUrlLarge"] stringValue]];
[book setCoverImage:[NSData
    dataWithContentsOfURL:imageURL]];

// Since there can be more than one author of a book, Authors are
stored in a child
// list of nodes. Fortunately, we now have a handy method for reducing
children to
// an array of their string values thanks to our category on NSXMLNode

NSXMLNode *authorsNode = [detailsNode
    childNamed:@"Authors"];
[book setAuthors:[NSMutableArray
    arrayWithArray:[authorsNode childrenAsStrings]]];

// We'll default the date last read to today, just to be user friendly

[book setDateRead:[NSDate date]];

// Okay, now our data model is populated... but how do we make the
interface
// reflect these changes? We call a method (next listing) that does it the
traditional
// Cocoa way

[self updateInterface];
}

// We allocated it, we're responsible for releasing it

[xmlDoc release];
}

```

Okay, so `doSearch:` gets the data, parses it, and populates our model. There's a missing link, however, which is to make the user interface reflect our data model. That's handled in a method cryptically called `updateInterface`, which simply copies the various values from our data model into the corresponding fields of the user interface, another tedious but straightforward bit of code writing.

But wait a second. It's not as straightforward as it first appears; there is potentially more than one author, and we've chosen an `NSTableView` for displaying them to the user. We can't just give an `NSTableView` an array to display. We need a table data source! For our simple application, we'll just implement the three required table data source methods right in `MTXMLAppDelegate` (which we earlier specified as the tables' data source in Interface Builder, you may recall). We mention these tasks here, without showing the code, to give a point of comparison for next month's article where we'll use Core Data

to eliminate most of these tedious, time-consuming aspects of building a Cocoa application.

Once we've taken care of updating the interfaces and implementing the table data source methods, we can compile and run our application. Type in the ISBN number from the back of a book you've got laying around, click **Lookup**, and it should automatically populate all of the fields except for date read and comments. If there's a problem with the ISBN, the error message will display in the lower left of the window.



Figure 4. The Compiled Application

Not much spit and polish, but it works! One task still remains, however, which is to write the code to save the data, with changes, to an XML file. Since we didn't retain the original data we retrieved from Amazon, we'll have to create XML from scratch.

Listing 3: MTXMLAppDelegate.m

saveToXML:

This method creates a new `NSXMLDocument`, populates it with the data from the user interface, then generates an XML file on disk from that data.

```

- (IBAction)saveToXML:(id)sender
{
    // Every XML document needs a root element, so we create a root
    element, then use it
    // to create our new document

    NSXMLElement *root = [[NSXMLElement alloc]
        initWithName:@"Book"];
    NSXMLDocument *doc = [[NSXMLDocument alloc]
        initWithRootElement:root];
    [root release];

    // We'll store the source URL where metadata should be stored: as an
    attribute rather
    // than as a child element

```



```

NSDictionary *attr = [NSDictionary
    dictionaryWithObject:[urlTF stringValue]
    forKey:@"url"];
[root setAttributesAsDictionary:attr];

// Now we'll store the strings

[root addChild:[NSXMLElement elementWithName:@"Comment"
    stringValue:[commentsTF stringValue]]];
[root addChild:[NSXMLElement elementWithName:@"Title"
    stringValue:[titleTF stringValue]]];
[root addChild:[NSXMLElement elementWithName:@"Publisher"
    stringValue:[publisherTF stringValue]]];
[root addChild:[NSXMLElement elementWithName:@"ISBN"
    stringValue:[isbnTF stringValue]]];

// We could add the sales rank as a string, but we'll do it as a number to
show how

NSXMLElement *rank = [NSXMLElement
    elementWithName:@"SalesRank"];
[rank setObjectValue:[NSNumber numberWithInt:
    [salesRankTF intValue]]];
[root addChild:rank];

// Dates can go in either as string or as NSDate, we'll store them as
NSDates

NSXMLElement *publishedDate = [NSXMLElement
    elementWithName:@"DatePublished"];
[publishedDate setObjectValue:
    [publishedDateDP dateValue]];
[root addChild:publishedDate];

NSXMLElement *readDate = [NSXMLElement
    elementWithName:@"DateRead"];
[readDate setObjectValue:[lastReadDP dateValue]];
[root addChild:readDate];

// We'll store the image's binary data. Letting the user change the image is
// beyond the scope of this article, so we'll just pull the image from our
data model
// object instead of the NSImageView.

NSXMLElement *cover = [[NSXMLElement alloc]
    initWithName:@"Cover"];
[cover setObjectValue:[book coverImage]];
[root addChild:cover];

// Up to now, we've used convenience class methods that return an
autorelease
// NSXMLElement. This time we allocated it ourselves to show how. Since
// we allocated it, we have to release it. This way is more memory
efficient because
// the object is released right away without going into the autorelease
pool. The
// downside is that we have to write more lines of code.

[cover release];

// Finally, the data in the author's table automatically gets updated in our
data model
// thanks to our table data source methods, so we'll pull this array right
from the data
// model as well

NSXMLElement *authors = [NSXMLElement
    elementWithName:@"Authors"];

int i;
for (i=0; i < [[book authors] count]; i++)
{

// Since we're in a loop of unknown size, we'll be kind and not use the
// autorelease pool.

```

```

NSXMLElement *author = [[NSXMLElement alloc]
    initWithName:@"Name"];
[author setStringValue:
    [[book authors] objectAtIndex:i]];
[authors addChild:author];
[author release];
}
[root addChild:authors];

// At this point, all of our data is now contained in our NSXMLDocument.
Let's write
// it to a file. The option NSXMLNodePrettyPrint tells it to use tabs and
other
// whitespace to make the output file easier on human eyes. You can pass
nil if you
// don't care how it looks

NSData *xmlData =
    [doc XMLDataWithOptions:NSXMLNodePrettyPrint];

// If this were a real app, we would present a sheet and ask where to
save. But this is a
// demo, so we're just going to dump the file onto the desktop.

[xmlData writeToFile:@"~/Desktop/book.xml"
    stringByExpandingTildeInPath atomically:YES];

// Memory cleanup
[doc release];
}

```

Now, let's compile and run again, plug in an ISBN and press the **Save** button. If all went according to plan, there will now be a file called **book.xml** on your desktop. Open this file up in a text editor or XML editor, and you should see our edited


UPS Ready



The Only Mac OS X Compatible UPS Shipping System

NRG Package Toolkit™ for UPS - offers a full set of tools that speed UPS functionality for shipping, rates and tracking parcels anywhere in the world.

Contact us today for a free Demo!

- Automated Tracking
- Address Validation
- Custom Laser Labels
- Thermal Printer and Digital Scale Support
- Easy to Integrate
- Works on MacOS X and Windows

NRG Software, LLC
P.O. Box 341338, Milwaukee, WI 53234-1338
262.363.4867 phone & fax

www.nrgsoftware.com



data as well-formed XML. Since we specified `NSXMLNodePrettyPrint`, it should appear in a human readable format with indents and line breaks, as you see in Figure 5 (although I've removed the binary image data from the XML in the screenshot).

```
<Book
  url="http://www.amazon.com/exec/obidos/ASIN/0974434469/
  1?dev-t=D34GBBJQXJFDS3%26camp=2025%26link_code=xm2">
  <Comment>I loved this book!</Comment>
  <Title>Mac OS X Technology Guide to Automator</Title>
  <Publisher>Spiderworks</Publisher>
  <ISBN>0974434469</ISBN>
  <SalesRank>71639</SalesRank>
  <DatePublished>2005-06-01T08:00:00</DatePublished>
  <DateRead>2005-06-03T20:01:14</DateRead>
  <Authors>
    <Name>Ben Waldie</Name>
  </Authors>
</Book>
```

Figure 5. Output XML

Bring It On Home

In this project, we've used NSXML to parse XML into Cocoa objects and to create XML from Cocoa objects. There's even more you can do with NSXML: You can delete nodes and edit nodes in place. You can create DTDs programmatically and validate existing XML against its DTD to make sure it conforms. From here, you should be able to figure out any XML-related task you need from Apple's developer documentation: NSXML is quite easy to use.

But speaking of "easy to use", you probably noticed that, in a few places, I mentioned certain aspects of building this Cocoa application that were not quite as developer-friendly as one would hope,

such as creating data model classes and synchronizing the user interface with the data model? Well, have no fear! Next month, we're going to dive into an even more amazing new technology: Core Data. We're going to recreate this application with more functionality in less time by writing less code.

References

"Namespaces in XML". W3C. World Wide Web Consortium. 6 June 2005. < <http://www.w3.org/TR/REC-xml-names/> >

"Introduction to Tree-Based XML Programming Guide for Cocoa". *Apple Developer Connection*. Apple Computer, Inc. 6 June 2005. < http://developer.apple.com/documentation/Cocoa/Conceptual/NSXML_Concepts/index.html >

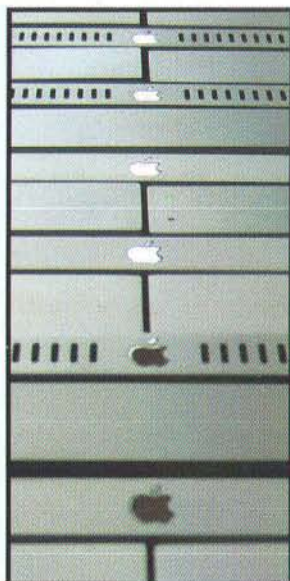
Accessorizer. Kevin Callahan. 6 June 2005. < <http://www.kevincallahan.org/software/accessorizer.html> >

"Amazon Web Services Documentation". *Amazon.com*. Amazon. 6 June 2005. < http://www.amazon.com/gp/browse.html/002-5739587-3639218?_encoding=UTF8&node=3487571 >



About The Author

Jeff LaMarche wrote his first line of code in Applesoft Basic on a Bell & Howell Apple //e in 1980 and he's owned at least one Apple computer at all times since. Though he currently makes his living consulting in the Mac-unfriendly world of "Enterprise" software, his Macs remain his first and greatest computer love. You can reach him at jeff_lamarche@mac.com.



Whoever said it's cheaper to stay home, didn't get out very often.

The cost of keeping servers in-house can far exceed the cost of server outsourcing.

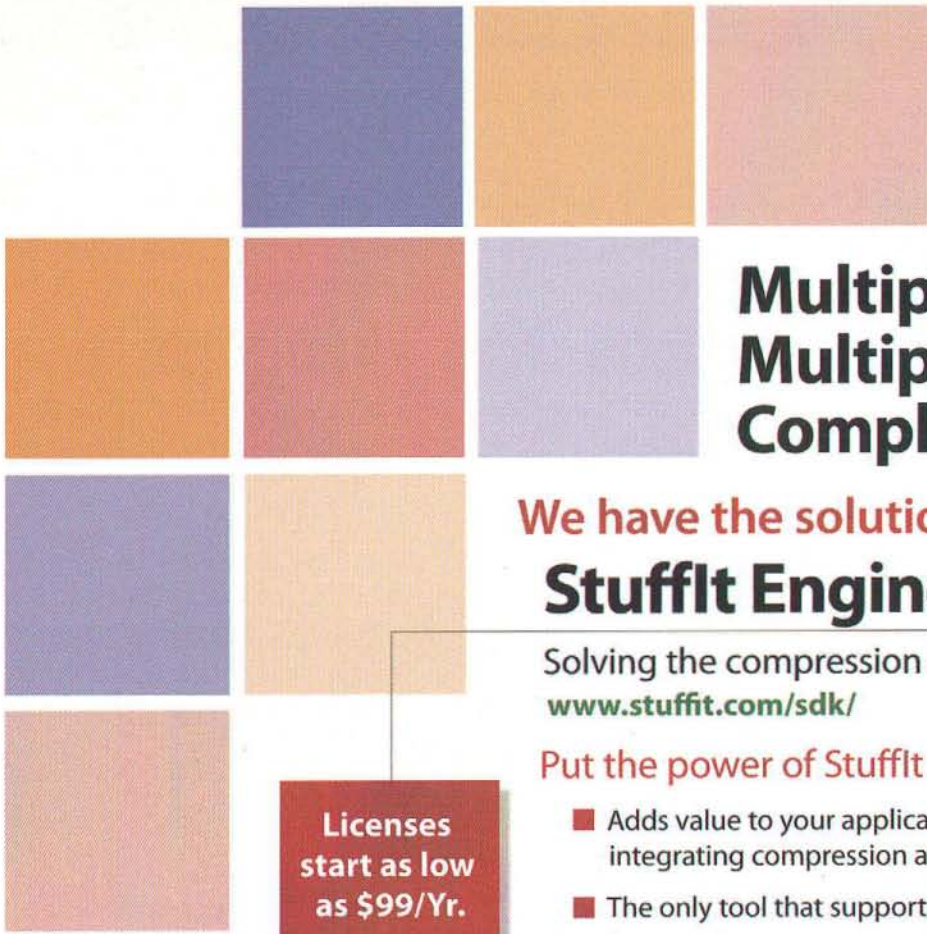
That's where XserveHosting comes in. With Xserve Colocation services, you can afford cost-effective, flexible, and highly reliable network and internet services, freeing up more of your time to take care of business. Don't stay home managing your servers – call us today at **949-480-9701** or visit our website at **xservhosting.com**.

Starting at only \$130.00 per month, Xserve Colocation includes:

- Free Setup for a savings of \$50.00*
 - Rackspace and Power
 - 1Mbit connection burstable to 100 Mbits with unlimited data transfer.
- * Please mention code MTM0305.

XSH
XSERVHOSTING
powering the xserve revolution

Xserve Colocation | Hosting | QuickTime Streaming



Multiple Formats. Multiple Platforms. Complex Installers.

We have the solution:

Stuffit Engine SDK

Solving the compression & multi-platform puzzle!

www.stuffit.com/sdk/

Put the power of Stuffit to work for you.

- Adds value to your applications by integrating compression and encryption tools.
- The only tool that supports the Stuffit file format.
- Make self extracting archives for Macintosh or Windows
- Available for Macintosh, Windows, Linux or Solaris



Licenses
start as low
as \$99/Yr.

Stuffit InstallerMaker

An OS X native version ready for developers!

www.stuffit.com/installermaker/

Give your software a solid beginning

- Create Macintosh OS X and Macintosh Classic compatible installers
- Get all the tools you need to install, uninstall or update your software in one complete package
- Add muscle to your installers by customizing your electric registration form to include surveys and special offers.

Prices
start at
\$250



www.allume.com

email: dev.sales@allume.com

2004 Allume Systems, Inc. Stuffit, Stuffit Installermaker and Stuffit Engine SDK are trademarks or registered trademarks of Allume Systems, Inc. The Allume logo is a registered trademark of Allume Systems. All other products are trademarks or registered trademarks of their respective holders. All rights reserved.

LAUNCHD: JUDGE, JURY, EXECUTIONER

THE CHANGES APPLE HAS WROUGHT

Apple, Apple, Apple. Where to start? We've all now heard about different kinds of transitions, and launchd marks a huge one. For a Unix-head that was comfortable with the systems that launchd consumes, I had to ask: why?

Introduction

My wife is an artist. When we were just dating, I bought her a set of markers that she had used before, and wanted for herself. As far as she was concerned, they were perfect: the angles of the nib, the size of the smaller side and the way they stacked up: perfect. She got about a year-and-a-half out of them. Naturally, markers run dry after use, and we needed to replace these markers. We went back to the art supply shop to find that the manufacturer had shipped new markers. Version 2-point-oh, if you will. Well, my wife disliked just about everything about the way these markers had changed. Change for change's sake. Is this the story with launchd?

Investigation

When I got my hands on Tiger during the beta period, I wanted to test how things in my then-current environment would work. I backup my PowerBook every night: a perfect job for cron. After clean-installing Tiger, and moving some data over, I went to go add my nightly backup to cron. What's this? A little note?

The periodic and atrun jobs have moved to launchd jobs
See /System/Library/LaunchDaemons

That piqued my interest. I had to follow that trail. Of course, during the beta period, you can't just post to your closest mailing-list asking, "what is this launchd thing?" Over in /System/Library/LaunchDaemons, I found the following file listing:

```
bootps.plist
com.apple.KernelEventAgent.plist
com.apple.atrun.plist
com.apple.mDNSResponder.plist
com.apple.nibindd.plist
com.apple.periodic-daily.plist
com.apple.periodic-monthly.plist
com.apple.periodic-weekly.plist
com.apple.portmap.plist
com.apple.syslogd.plist
com.apple.xgridagentd.plist
com.apple.xgridcontrollerd.plist
com.vix.cron.plist
comsat.plist
eppc.plist
exec.plist
finger.plist
ftp.plist
login.plist
nmbd.plist
ntalk.plist
org.isc.named.plist
org.postfix.master.plist
org.xinetd.xinetd.plist
printer.plist
shell.plist
smbd.plist
ssh.plist
swat.plist
telnet.plist
tftp.plist
```


Looks like familiar stuff, but presented in a new way. Is there a man page for this thing? Sure enough, there is. The man page says simply that launchd is the "System wide and per-user daemon/agent manager". A little further down, it hits me: "During boot **launchd** is invoked by the kernel to run as the first process on the system and to further bootstrap the rest of the system."

Kaboom

Run as the first process? Don't be silly, that's initd's job! Look, I'll just go find initd in my process list and...hmmmm. No initd. OK...breathe slowly.

First cron, and now initd. This launchd thing is ravenous! What else has it taken over? As it turns out, launchd now takes the place of several tried and true systems, namely:

init
Startup Items
inetd/xinetd
watchdog (on OS X Server)
cron
rc (almost)

The rc structure still exists, but there is a note in the (very short) launchd man page, that states, "At some point in the future, we hope to completely phase out the use of **rc**." Six time-honored, tried-and-true subsystems that get the job done, and are in use on other systems. Again, it makes me ask "why?"

Why

Why take these well-known subsystems and replace them with something unknown and untested? One thing I thought I might find the answer to while at WWDC this year was the reason for the change. Turns out that there was nary a mention of launchd in all of the sessions I attended. There was so much other good content, that I didn't think about it all too much while I was there, but then I got home and looked at all of the non-WWDC e-mail that had accumulated during my trip. I had three requests all looking for more information relating to launchd (though, they just didn't know it yet). While I still don't have the exact answer to "why?", I have been using launchd and it's related programs for various tasks, and it seems to work reasonably well.

Apple's OS X web page (<http://www.apple.com/macosx/features/unix/>) makes a mention of launchd thusly:

"Since Mac OS X rarely requires you to reboot, you'll hardly ever notice it, but Tiger takes less time to start up, thanks to launchd. Launchd provides faster startup through a unified framework for starting, stopping and managing daemons, and incorporates inetd,

init, mach_init, System Starter and related services. Administrators have a single mechanism for auditing, configuring and setting resources limits on services."

Well, OK, that explains a little bit...or perhaps even more than a bit. Look at what's happening here. Instead of this (10.3):

```
$ ps ax
PID  TT  STAT      TIME COMMAND
  1  ??  Ss       0:00.85 /sbin/init
  2  ??  Ss       5:35.22 /sbin/mach_init
...
```

Tiger gives us this:

```
PID  TT  STAT      TIME COMMAND
  1  ??  S<s      0:04.40 /sbin/launchd
...
```

launchd is now responsible for pretty much everything. All your bases belong to launchd!

P to the L to the I-S-T

When you read the man page for launchd, you may notice under the 'see also' section, that it mentions launchd.plist. If you go looking on your system for launchd.plist, you're not going to find it. They're actually referring to the *concept* of launchd plists.

For the uninitiated, 'plist' stands for "Property List." Plists provide a standardized way for applications to store data, and then later retrieve this data. You can think of it like a better Windows INI file. However, there are three kinds of plists: old-style ASCII, XML and binary. The ASCII format has been inherited from NeXT, and still supported for legacy reasons. The binary version was introduced with 10.2, but has only really been pushed by Apple in Tiger. launchd stores its plists in XML format. XML plists are plain-text files that can be read by a text editor. Even better is to use an editor that understands plists. Apple's GUI-based Property List editor is one such editor. I also hope to have a curses based (TUI) editor finished by the time this goes to print (life, however, may get in the way of that deadline). So, for now, copy /System/Library/LaunchDaemons/ssh.plist to your desktop or some other out of the way place, and open it with "Property List Editor" (which, under Tiger is in your Applications directory. Panther users need to have the developer tools installed, and will find it under /Developer/Applications/Utilities, however, if you're on Panther, you don't have launchd...but it's a nice exercise. A double-click on the plist should launch Property List Editor also, if installed). Figure 1 illustrates how the hierarchal key/value system is displayed.

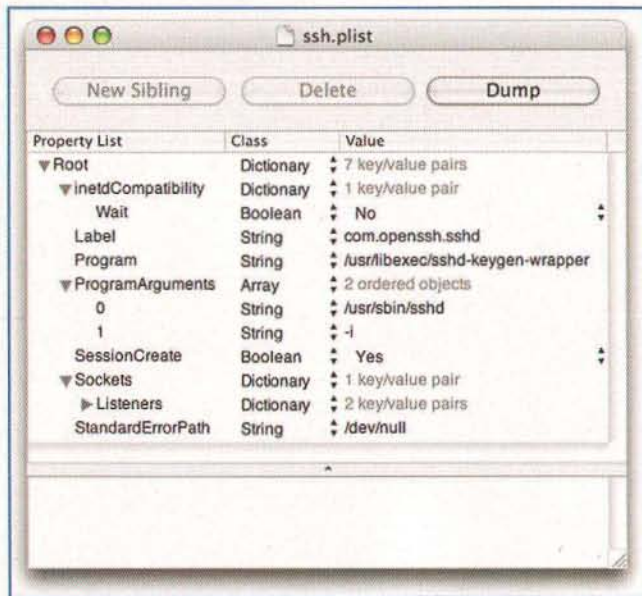


Figure 1. Property List Editor displaying sshd.plist

Property lists are able to store several Core Foundation types (CFString, CFNumber, CFBoolean, CFDate, CFData, CFArray, and CFDictionary), and are really aimed at storing values of 100kb or less. You'll notice that all of the types are basically strings and numbers. Also, in addition to simple flat list of key/value pairs, parameters in a plist can be structured hierarchically. So, what are the parameters of a launchd plist, and how do we write our own? Right this way, my friend...

DIY

The first thing to notice from Figure 1 is the 'Label' key. This is a way to uniquely identify a job to launchd. You'll see a little later on why launchd needs to do this. 'Label' can be any value you wish, however, it's easiest to name it something resembling the job, or stick with the domain scheme Apple has adopted (yes, 'adopted', not 'created'). The next thing to do is to specify the program that you want to run. Interestingly (or frustratingly), there are two ways to specify this. One way, like the example in Figure 1, is to use the 'Program' key, and specify the value as the full path to the executable. If the program has any parameters, those can be specified in the 'ProgramArguments' array key. The second way to specify the program to run is to forgo the 'Program' key altogether, and simply use 'ProgramArguments'. Using this method, the first value in the array is taken as the executable, and the remaining array elements are the arguments. Let's make a sample plist. First, we need something to run. How about a script that speaks the current time? Here it is:

```
$ mkdir bin
$ cd bin
~/bin $ vi speakdate.sh
#!/bin/bash
say `date "+%H:%M"`
:wq
```

(note that this is in my home directory, and I'm running as a

regular user, not root)
Make sure it's executable:

```
~/bin $ chmod 750 speakdate.sh
```

...and test it:

```
$ ./speakdate.sh
```

You should hear the current time (in 24-hour 'military' designation) spoken. Now we can create our plist for launchd. I'm going to recommend using Apple's Property List Editor for the simple fact that you'll get all of the DTD tag info right, and the editor will write proper XML without you breaking a sweat. We'll look at it after the editor gets through with it.

Once the editor is up, click on the 'New Root' button. Knock the disclosure triangle down on the newly created root node, and then click on the 'New Child' button three times. Change the key names, classes and values to the following values:

Name: Label
Class: String
Value: speakdate

Name: OnDemand
Class: Boolean
Value: Yes

Name: ProgramArguments
Class: Array

Once again, highlight this array, drop the disclosure triangle and click on the 'New Child' button in the toolbar. Simply change the value of item 0 in this array to be the full path to the script – mine's at /Users/marczak/bin/speakdate.sh and your's will vary based on your user name.

Save this plist as '~/.Library/LaunchAgents/speakdate.plist'. The final plist, as seen in Property List Editor is shown in Figure 2. (Please note that like everything else on OS X, you should follow the system hierarchy: per user files go in ~/.Library/Launch*, system-wide additions go into /Library/Launch*, and you should never touch /System/* !)

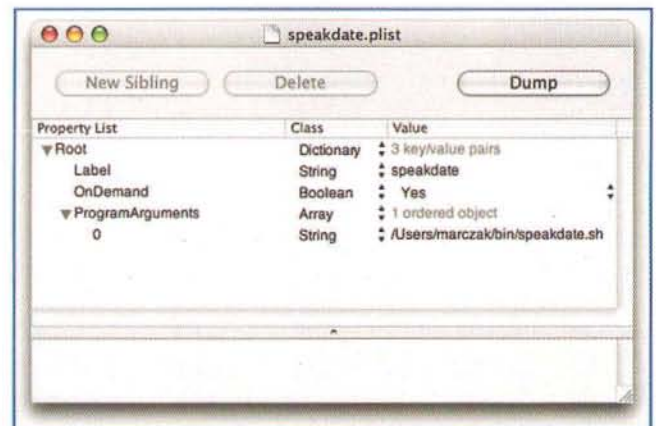


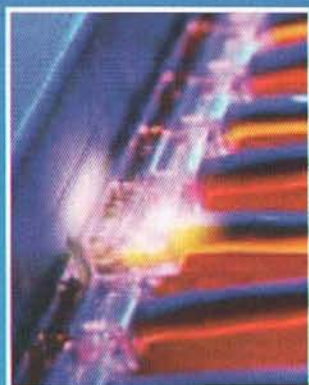
Figure 2: Our plist for launchd, version 1

Data Banks Communications Inc

*HOME OF THE FREE COMPUTER
THAT'S RIGHT FREE COMPUTER!*

**GET A MAC MINI OR DELL
COMPUTER FREE JUST SIGN UP
FOR VOIP DSL BUNDLE**

INFOSPEED 768K/128K	\$ 69.95
INFOSPEED 1.5M/128K	\$ 69.95
INFOSPEED 1.5M/384K	\$ 69.95
INFOSPEED 7.1M/768K	\$ 199.95
INFOSPEED 384K/384K	\$ 69.95
INFOSPEED 768K/768K	\$ 149.95



TeleSoho 1.5M/128	\$69.95
TeleSurfer Plus 608K/128K	\$69.95
Residential 768K/128K	\$69.95
SOHO1.5M/768K	\$79.95
SOHO3.0M/384K	\$79.95
SOHO3.0M/768K	\$79.95
SOHO6.0M/768K	\$139.95
TeleSpeed 144K/144K	\$125.95
TeleSpeed 192K/192K	\$125.95
TeleSpeed 384K/384K	\$149.95
TeleSpeed 768K/768K	\$159.95
TeleSpeed 1.1M/1.1M	\$199.95
TeleSpeed 1.5M/1.5M	\$249.95

Ask for business line special!

www.databanksglobal.com

For questions regarding sales or service, please call
us 24 hours a day at 1-866-624-6114

Just for completeness sake, let's see what the file *really* looks like:

```
$ cat speakdate.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>speakdate</string>
  <key>OnDemand</key>
  <true/>
  <key>ProgramArguments</key>
  <array>
    <string>/Users/marczak/bin/speakdate.sh</string>
  </array>
</dict>
</plist>
```

Again, we see that by using an editor that understands the plist format, a lot gets filled in for us. Of course, one could do this by hand, but you risk more errors that way.

How do we control this launchd thing? By using a CLI utility named 'launchctl'. launchctl can do many things, but for now, let's look at loading a plist into its environment. To load speakdate, issue the following command:

```
launchctl load -w
/Users/marczak/Library/LaunchAgents/speakdate.plist
```

The 'load' verb instructs launchctl to tell launchd to add the plist to its list of jobs. The '-w' flag removes the disabled flag from the plist, and updates the plist itself. We can see that it has been added to our per user list by using launchctl with the 'list' verb:

```
$ launchctl list
speakdate
```

It's really beautiful that launchd operates on a per user basis. It wouldn't surprise me if we see launchd taking over user login items in the future. Now, on to the magic – making it run! The 'start' verb tells launchctl to have launchd start a job. The second parameter is the label you gave the job in the plist. It's as easy as:

```
$ launchctl start speakdate
```

In a second or two, your Mac should speak the current time (make sure your volume is up!). All in all, that was pretty lame, right? For all that work, you could manually run the script and have your Mac speak the time. Faithful readers of this column know that I'm all about automation, so let's get to it. First, we'll unload the plist:

```
launchctl unload -w
/Users/marczak/Library/LaunchAgents/speakdate.plist
```

The 'unload' verb does what you expect. Like the 'load' verb, there's a '-w' flag. This time, it marks the plist as *disabled*, and writes that flag into the plist. Before we run Property List Editor, we need a test folder for this example. Create /Users/Shared/watch. Now bring up Property List Editor and open up our plist (use the handy 'File->Open Recent' menu for quick access. See? Sometimes it's the little things that count!).

Tilt down the disclosure triangle on 'Root', then click the 'New Child' button. Name the key 'WatchPaths', make it an array, and then click 'New Child'. Change the value for item 0 to '/Users/Shared/watch'. Save it, load it, and invoke it:

```
$ launchctl unload
/Users/marczak/Library/LaunchAgents/speakdate.plist
$ launchctl load -w
/Users/marczak/Library/LaunchAgents/speakdate.plist
$ touch /Users/Shared/watch/temp1
```

Tell me you don't think that's pretty cool! Really, go on. Have fun and do that again:

```
$ touch /Users/Shared/watch/temp2
```

The value for "WatchPaths" can also be brought down to the file level (watching a particular file). Another way to watch an entire directory is with the use of the "QueueDirectories". It differs from "WatchPaths" by running the program until the queue is cleared. I'll illustrate this with our example, once again. Unload speakdate, first. Load up the plist into Property Editor. Change the "WatchPaths" key name to "QueueDirectories". Save, quit and load it with launchctl. You'll notice that...uh...hello? You can stop now. Really. Please, stop telling me the time. I'm begging you! The quick and easy way to keep the script loaded, but make it idle is to clear the queue directory:

```
$ rm /Users/Shared/watch/*
```

Ahhhh, peace and quiet. "QueueDirectories" is a great way to have a script process items that come into a directory and need to be dealt with. Of course, you'll need to move or remove them once you've processed them.

Just like cron! Kinda...

As mentioned, in Tiger, launchd has taken the place of cron...mostly. cron does still exist and will run jobs that you put in the various crontab files (thankfully – see "Utopia, it's not", below). If you want to do it the launchd way, there are two separate keys that will help. "StartInterval" takes a numeric value ("n"), in seconds, and causes the job to fire every *n* seconds. This one's simple. So simple, that I even want to introduce an additional key: "RunAtLoad". "RunAtLoad" takes a boolean that determines if the program runs once when loaded, or not. Just so this does what you expect, we'll use it in our example. Do the 'launchctl unload' and edit dance once again. Delete the "QueueDirectories" node. Add a new Boolean key called "RunAtLoad" and make sure its value is 'yes'. Add another key, this time numeric, called "StartInterval". Make the value 10. Save it, and go load it. You should immediately hear the time spoken, and then every ten seconds after. StartInterval: says what it does, does what it says. Unload it when you've heard enough.

According to the man page, the "StartCalendarInterval" key is supposed to cause "the job to be started every calendar interval as specified. Missing arguments are considered to be wildcard. The semantics are much like crontab". Well, it's

supposed to – please see “Utopia, it’s not,” below. Let’s run our example again. Edit the plist once again.

Delete the “StartInterval” key, and create a new one named “StartCalendarInterval”. Make its class a dictionary, and then add two children. Each child should be a number. Name the first child key ‘Hour’ and the second ‘Minute’. ‘Hour’ is the hour number, from 0 to 23, and minute is the minute, from 0 though 59. Just so you can hear this job run, choose something in about 5 minutes from the current time. I want my list to run at 8:30pm, as you can see in figure 3.



Figure 3. Our sample plist, now using <StartCalendarInterval>

Save the new version, and then use `launchctl` to load it up. When the clock strikes the appropriate hour and minute, the script will run and you’ll hear the time spoken. Any values that are left out, are considered cron-like wildcards. In our case, this script is intended to run at 8:30pm every day, of every month.

Didn’t you mention xinetd?

Yes, `launchd` also takes over for `xinetd`, and there’s a whole host of ways that it stays compatible. The interesting thing is the way `launchd` handles this. In Apple’s quest to speed up boot times, `launchd` basically fakes the fact that there’s a bunch of programs to start, and it doesn’t really start any of the ones that would traditionally be started. It does read the plists and figure out which ports to reserve and listen to.

Since you’ve probably had enough editing of plists for now, let’s look at two services that Apple had been running through `xinetd`, and how they made the transition to `launchd`. The first of those being `sshd` (which, on other platforms I’ve dealt with, usually runs standalone), and the second, `ftpd`.

We saw the plist for `sshd` back in figure 1, and there’s a lot going on in there! Try looking at the file itself:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
```

```
<key>Label</key>
<string>com.openssh.sshd</string>
<key>Program</key>
<string>/usr/libexec/ssh-keygen-wrapper</string>
<key>ProgramArguments</key>
<array>
  <string>/usr/sbin/sshd</string>
  <string>-i</string>
</array>
<key>SessionCreate</key>
<true/>
<key>Sockets</key>
<dict>
  <key>Listeners</key>
  <dict>
    <key>Bonjour</key>
    <array>
      <string>ssh</string>
      <string>sftp-ssh</string>
    </array>
    <key>SockServiceName</key>
    <string>ssh</string>
  </dict>
</dict>
<key>StandardErrorPath</key>
<string>/dev/null</string>
<key>inetdCompatibility</key>
<dict>
  <key>Wait</key>
  <false/>
</dict>
</dict>
</plist>
```

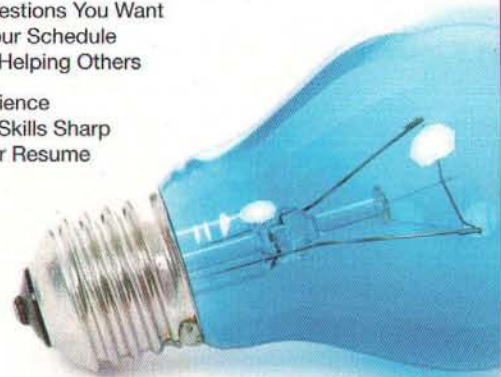
You’ve already seen ‘Label’, ‘Program’ and ‘ProgramArguments’. Here’s the goodies: “SessionCreate”, “Sockets” and “inetdCompatibility”.

“inetdCompatibility” is really the thrust of this whole thing, so I’ll start there. This tells `launchd` that this program thinks that it’s being run though `inetd/xinetd`. The ‘wait’ subkey is the analog of `xinetd`’s wait parameter. Basically, we need to know if this program is single or multi-threaded. If it’s

Join Our Support Team.

Join our team of volunteers providing free personalized technical support covering hardware, software, networking and other Mac related issues.

- Answer Questions You Want
- Work on Your Schedule
- Feel Great Helping Others
- Gain Experience
- Keep Your Skills Sharp
- Add to Your Resume



macosX.com

The Answer to Mac Support

single-threaded, you want to set 'Wait' to true. This way, launchd runs the program, and then stops answering for it. If set to 'false', launchd will continue to accept connections for the program, and spawn new copies as needed. Somewhat undocumented is the fact that a separate program, 'launchproxy', handles programs that need inetd compatibility. 'launchproxy' has a short little man page, but isn't mentioned in launchd's other man pages.

"SessionCreate" is also mostly undocumented. Undocumented as far as the man page, Apple Developer docs and that little thing called Google. Of course, there's always the source. Defining this key reaches into Apple's security framework and creates a security session for the launched program. Now, this only works if "inetdCompatibility" is not configured to wait.

"Sockets" is an optional key that tells launchd to use on demand sockets to trigger the job. The job can check-in and get a copy of the file descriptors in use. There are several sub-keys that go along with "Sockets", such as:

Listeners/SockServiceName – Important one: determines the service that the program 'bind's or 'connect's to, based on /etc/services. (This can get a little frustrating. For example, if you want to run ssh on an alternate port, you need to add it in /etc/services, and then tell the plist that the "SockServiceName" should bind to that alternate name. Goofy).

Listeners/Bonjour – This is an incredibly easy way to register the service with the mDNSResponder. Brilliant.

Quick example: on a server, look at the process list before anyone tries to access it over ssh:

```
# ps ax | grep ssh
15922 p1 S+ 0:00.01 grep ssh
```

No permanent sshd is running. Once someone gets in via ssh:

```
# ps ax | grep [s]sh
15939 ?? S 0:00.11 /usr/sbin/sshd -i
```

launchd is doing its job. As promised, here's ftp's launchd.plist:

```
# cat /System/Library/LaunchDaemons/ftp.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer/DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Disabled</key>
  <true/>
  <key>Label</key>
  <string>com.apple.xftpd</string>
  <key>Program</key>
  <string>/usr/libexec/xftpd</string>
  <key>ProgramArguments</key>
  <array>
    <string>xftpd</string>
    <string>-a</string>
  </array>
  <key>Sockets</key>
  <dict>
    <key>Listeners</key>
    <dict>
```

```
      <key>SockPassive</key>
      <true/>
      <key>SockServiceName</key>
      <string>ftp</string>
      <key>SockType</key>
      <string>SOCK_STREAM</string>
    </dict>
  </dict>
  <key>inetdCompatibility</key>
  <dict>
    <key>Wait</key>
    <false/>
  </dict>
</dict>
</plist>
```

The key differences from sshd's plist are the keys "SockPassive" and "SockType":

SockPassive – Is this a 'listen' socket, or connect? Default/true is 'listen'.

SockType – what kind of socket to create (stream – the default, dgram and seqpacket).

Again, these are part of the "Sockets" key.

Avertissement !

There are certain ways that launchd expects a daemon to behave. Specifically (and this is right from the man page):

A daemon or agent launched by launchd MUST NOT do the following in the process directly launched by launchd:

- fork(2) and have the parent process exit(3) or _exit(2).

- Call daemon(3)

A daemon or agent launched by launchd SHOULD NOT do the following as a part of their startup initialization:

- Setup the user ID or group ID.
- Setup the working directory.
- chroot(2)
- setsid(2)
- Close "stray" file descriptors.
- Change stdio(3) to /dev/null.
- Setup resource limits with setrusage(2).
- Setup priority with setpriority(2).
- Ignore the SIGTERM signal.

A daemon or agent launched by launchd SHOULD:

- Launch on demand given criteria specified in the XML property list. More information can be found in launch(3).
- Catch the SIGTERM signal.

Those are some pretty important guidelines. Mind them as you're writing your scripts!

Utopia, it's not

While launchd has a lot of great features, it can still be a little rough around the edges, somewhat typical of a 1-dot-oh release. In particular:

Sometimes, nothing happens. I've had problems with launchd that, after loading a plist, I can't get the job to start. However, after a reboot, the same plist runs just fine. Can't explain that one just yet.

Sometimes, something happens, just not what I expect. Again, a reboot seems to fix it. Case in point: while testing the plist for this article, I wrote the sample and loaded it. I could start it just fine. But asking for a list gave me:

```
com.apple.launchd.firstborn: execve(): No such file or directory
```

...and then launchctl needed to be killed to get my prompt back. After a reboot, the same, untouched plist loaded and listed just fine.

It's the intended replacement for cron, however, there are two problems:

It doesn't fully encompass everything cron can do. Such as run a command in a limited scope, such as every 3 hours, Monday through Friday, only between 8am and 8pm ("0 8-20/3 * * 1-5 root /path/to/program", for example.)

<StartCalendarInterval> doesn't work! Well, it does...once. After which, it never fires again. Not fixed in 10.4.1. (hopefully, 10.4.2 nails this issue down).

No graceful way to unload a daemon. Now, some hardcore Unix folks will yell at me about this one. That's just how some Unicities work. OS X has SystemStarter to deal with this, and it's still in place. I imagine that launchd will someday overtake this system as well. However, if you have services that need to be started or stopped a very specific way, you may find that writing a StartupItem is, for now, the way to go.

The workarounds for now seem to be the use of the 'old' systems: cron and SystemStarter. If you need some very specific way to shut down a process (fmserv users, I'm looking in your direction), make it a StartupItem. I have OpenBase running on some Tiger systems this way. Works just fine.

As for StartCalendarInterval, well, just use cron. You can simply copy over any crontab entries you had been using on Panther, or create new entries, and put them in the appropriate place. Apple was kind enough to make sure that launchd calls cron to run any jobs that it should. Of course, you can use launchd to unload and re-load cron:

```
# launchctl unload
/System/Library/LaunchDaemons/com.vix.cron.plist
# launchctl load
/System/Library/LaunchDaemons/com.vix.cron.plist
```

cron also remains the more flexible of the two in terms of scheduling flexibility. That's life on the edge, I suppose.

What about watchdog?

Yes, I did mention that launchd replaces watchdog. This takes place thanks to the "OnDemand" key. When this Boolean

Long Distance

2.9¢ Per Minute!

Straight 6 second billing increments

Excellent rates on intrastate, intralata/toll calls and international calling with no term contract.

Toll Free (800/888/877/866) service, same low per minute rate for incoming calls.

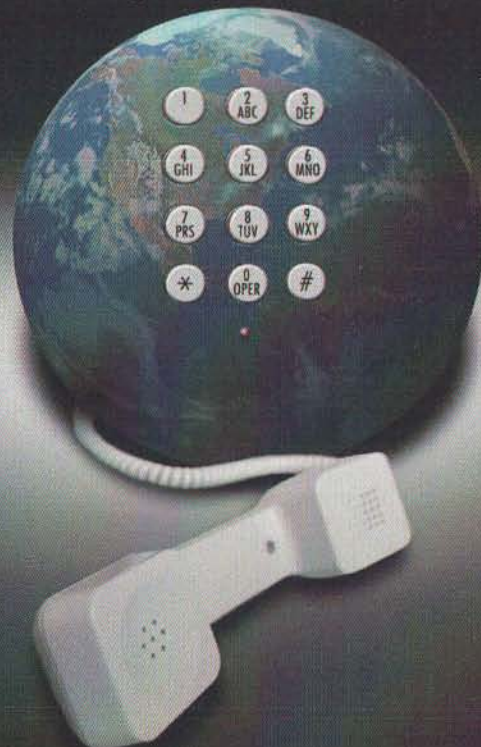
10 cents per minute calling card.

Detailed billing directly from OPEX.

Quality electronic and telephone customer support.

No monthly billing fee if your bill is over \$20.00 each month.

(NOTE: \$2.00 billing fee is charged when your bill is under \$20.00.)



key is set to 'false', launchd watches the process and restarts it if necessary. Try this plist:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>iCalForever</string>
  <key>OnDemand</key>
  <false/>
  <key>ProgramArguments</key>
  <array>
    <string>/Applications/iCal.app/Contents/MacOS/iCal</string>
  </array>
</dict>
</plist>
```

Load it into launchd:

```
$ launchctl load
/Users/marczak/Library/LaunchAgents/icalforever.plist
```

There's iCal! Now, try to quit it. Back it comes!...and again tell it to quit...there it is again! Unload the job to rid yourself of iCal for a bit:

```
$ launchctl unload
/Users/marczak/Library/LaunchAgents/icalforever.plist
```

If you really were quitting it quickly, and you still had your terminal visible, you'll notice a series of messages:

```
launchd[1447]: iCalForever: exited: Terminated
launchd[1447]: iCalForever: respawning too quickly! throttling
launchd[1447]: iCalForever: 9 more failures without living at
least 60 seconds will cause job removal
launchd[1447]: iCalForever: will restart in 10 seconds
```

This is really nice protection against some job going bonkers and taking down your system.

Conclusion

So, is launchd really like my wife's situation with her illustration markers? Is it really just change for change's sake? In this case, the answer is no. Just as the Unix world went through

the transition from stand-alone daemons to inetd, and ultimately xinetd, we're ready for launchd. The real question is if launchd is ready for us. Fortunately, cron and xinetd are still in place and are called by launchd. cron continues to do the right (and more flexible) thing, so use it if you need to. Launchd is *extremely* version one-dot-oh, but I expect that it will come up to speed very quickly. All in all, launchd has the right motivation: a unified way of launching and managing daemons, no matter which stage of system operation they're run at.

I think that most people's gripe with launchd is simply the scarcity of documentation. The Tiger server docs briefly mention launchd, and even in those cases refer you to the man page. Actually, that's something that even I haven't done yet! If you do want to dig into launchd, you should read the man pages. While you will still be left with some questions, it does get you going.

The "Utopia it's not" section above lists a few workarounds to launchd's bugs. Even in the source code, there's a TODO list. One of the items on the list? "cron". Another? "Documentation". Indeed.

I was just about to package this month's column up, and get it in for print when I was made aware of a plist editor *specifically* for launchd items. I haven't used it, so I can't vouch for it, but it looks nice. It's called (amazingly), "Launchd Editor", and can be found at <<http://www.codepoetry.net/products/launchdeditor>>. Worth a mention.

Dig in and experiment with launchd – it's probably the best way to learn its many ins and outs. I'll certainly be following the development of launchd pretty closely, and will be updating you as I know more.

MM

About The Author



Ed Marczak, owns and operates Radiotope, a technology consulting company that tries its hardest to avoid bad puns regarding launchd, such as 'time for launch' or '3, 2, 1, launch!'. Get launched at <http://www.radiotope.com>

Joining Worlds of Information

Valentina 2: Deploy True, Royalty Free Client-Server Solutions

www.paradigmasoft.com

PARADIGMA SOFTWARE



DevDepot has it all!

Get More out of your Mac!

Visit our online store today for special offers and great new products.
www.devdepot.com

KEYSPAN

Digital Media Remote

Control your computer's media programs just like your TV!

For iTunes, Windows Media Player, DVD, CD, and more!



ONLY
\$39⁹⁹

iTripmini

FM Transmitter

Listen in your car!

Designed exclusively for the iPod mini.



NO BATTERIES NEEDED!

ONLY
\$39⁹⁹

iTalk

iPod Voice Recorder

Works as a loud speaker!

Turn your iPod into a world-class voice recorder.



ONLY
\$34⁹⁹

SightLight

FireWire light for iSight

\$37⁹⁹



AC Adapter

For Powerbook and iBook laptops

\$39⁹⁹



Keypoint Remote

Multimedia RF remote control

\$49⁹⁹



Noise Reduction Headphones

\$59⁹⁹



iMic

USB Audio Interface

A must-have device for people who are serious about high quality audio.

Connect virtually any sound device!



ONLY
\$34⁹⁹

FlashDrive

Cyclops 256 MB USB Flash Drive

\$79⁹⁹



PowerWave

USB Audio Interface & Amplifier

\$89⁹⁹



PowerMate

USB Multimedia Controller & Input Device

\$36⁹⁹



iceKey

Slim USB Keyboard

\$44⁹⁹



TechTool Pro 4.0 for Mac OS X

The ultimate emergency software is now available for OS X!

\$89⁹⁹



UNIX Utilities for Mac OS X 3.0

Everything you need to turn your Mac into a UNIX Workstation.

\$39⁹⁹



Office Applications for Mac OS X 3.0

Packed with office and productivity apps!

\$29⁹⁹



Office Applications for Mac OS X 3.0

Bug reporting and organizing!

* Includes free bug reporter with each application release!



\$79⁹⁹



DevDepot is not responsible for typographical errors. Offers subject to change at any time. © 1984-2004 Developer Depot, Inc. Some material copyright of their respective holders. All Rights Reserved. Developer Depot, Inc. is a division of Allume Systems, Inc. located in California.

www.devdepot.com

Advertiser/Product Index

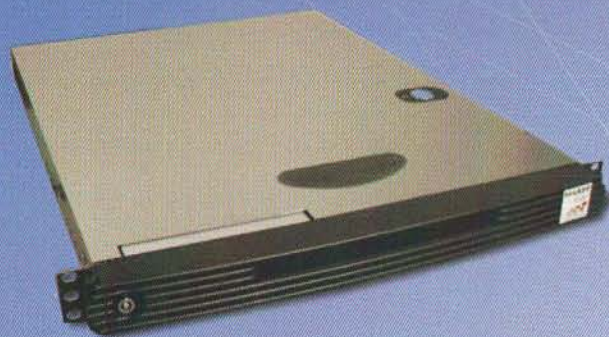
Aladdin Knowledge Systems, Inc.	BC
Allume Systems, Inc.	69
BetterRAM.com •	17
Big Nerd Ranch, Inc.	51
Data Banks Communications.	19
DevDepot	42-43
DevDepot	31
Digital Crowd, Inc.	75
Excel Software	5
FairCom Corporation	1
InterSystems Corporation.	41
John Wiley & Sons	35
Kerio Technologies, Inc.	34
MacDirectory	23
NRG Software, LLC	67
OpenBase International, Ltd.	39
Paradigma Software	74
Portlock Software	76
Seapine Software, Inc.	7
SharpNET Solutions, Inc.	53
Small Dog Electronics	13
SmileOnMyMac, LLC	29
Spiderworks	IFC, 50
TOLIS Group, Inc.	IBC
Utilities4Less.com	77
XSH Hosting LLC	68

Big Nerd Ranch • Big Nerd Ranch, Inc.	51
bruAPP Network Backup System • Tolis Group Inc.	IBC
c-tree Plus • FairCom Corporation	1
Cache • InterSystems Corporation	41
Digital Rights Management • Aladdin Knowledge Systems, Inc.	BC
Extreme Macs • John Wiley & Sons	35
InstallerMaker, StuffIt • Allume Systems	79
Internet Marketing Services • SharpNET Solutions, INC.	53
Kerio Mail Server • Kerio Technologies, Inc.	34
Long Distance Phone Service • Utilities4Less.com	77
Mac A&D • Excel Software	5
MacDirectory Magazine • MacDirectory	23
MacOSX.com • Digital Crowd, Inc.	75
Maximizing Your Mac! • DevDepot	42-43
NRG Package Toolkit For UPS • NRG Software, LLC	67
OpenBase • OpenBase International, Ltd.	39
PDF Pen • SmileOnMyMac, LLC	29
Portlock Storage Manager • Portluck Software	76
Ram • BetterRAM.com	17
SmallDog Electronics • Small Dog	13
SpiderWorks ebooks • Spiderworks	IFC, 50
Test Track Pro • Seapine Software, Inc.	7
Tools • DevDepot	31
Valentina • Paradigma Software	74
VOIP • Data Banks Communications	19
XSERVHOSTING • XSH Hosting LLC	68

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

Introducing the bruAPP™

Plug & Play Mac OS X Network Backup System from TOLIS Group



- D2D + D2D2T Backup Appliance
- 250 GB - 4 TB disk stage **NEW**
- Scalable to support future network growth
- Remote management via graphical and text consoles

- Fully compatible with Xsan environments
- Support for all SCSI and Fibre-Channel tape devices and libraries
- Built in VXA-2 or LTO-2 tape drives in select models
- Client system support for all major operating systems
- BRU™— 20 years of Backup You Can Trust™



One System, One Function, One Solid Performer
5 Minutes From Unpack to First Backup—Literally

Whether you're responsible for a few desktide systems or the latest Hollywood blockbuster, the new bruAPP provides ultra-reliable, easy to use disk-to-disk (D2D) or disk-to-disk-to-tape (D2D2T) network-based backup.

bruAPP complements TOLIS Group's BRU Server for Mac OS X and BRU LE for Mac OS X data backup and restore software products. bruAPP pricing starts at \$2,999.

To learn more about the bruAPP and BRU products for Mac OS X, please call 480-505-0488 ext. 252 or visit TOLIS Group on the web at www.tolisgroup.com.



“HASP...was the only Software Digital Rights Management solution we would consider... incredibly strong security, so easy to use.”



Pierre Maloka DIRECTOR OF GAME DEVELOPMENT
Mel Liu SOFTWARE TECHNICIAN

⊕ TLC Industries is the company behind FlexArcade™, the revolutionary game platform. PowerBall Billiards™, TLC's pool simulation, is quickly becoming an arcade favorite. When software is this hot, it has to be protected by the best. To enable safe, secure delivery to customers, TLC chose HASP, the world's #1 hardware-based software security solution*.

As TLC's Maloka and Liu know, the HASP Envelope offers more layers of protection than any software security solution, making it the strongest file wrapping technology available. And the user-friendly interface makes the solution virtually **plug-and-play**. This means TLC can concentrate on developing great games—and leave the security to the security experts.

*IDC Bulletin #31432, 2004.

Visit Aladdin.com/hasp to view HASP white papers, an online demo, or to request a Developer Evaluation Kit.

For information about TLC or PowerBall Billiards, visit tlcind.com.

